

Allen-Bradley

Logix5000 Controllers Common Procedures

**1756 ControlLogix,
1769 CompactLogix,
1789 SoftLogix,
1794 FlexLogix, and
PowerFlex 700S with DriveLogix**

Programming Manual

Rockwell Automation

Important User Information

Solid state equipment has operational characteristics differing from those of electromechanical equipment. Safety Guidelines for the Application, Installation and Maintenance of Solid State Controls, publication SGI-1.1 available from your local Rockwell Automation sales office or online at: http://literature.rockwellautomation.com describes some important differences between solid state equipment and hard-wired electromechanical devices. Because of this difference, and also because of the wide variety of uses for solid state equipment, all persons responsible for applying this equipment must satisfy themselves that each intended application of this equipment is acceptable.

In no event will Rockwell Automation, Inc. be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variables and requirements associated with any particular installation, Rockwell Automation, Inc. cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by Rockwell Automation, Inc. with respect to use of information, circuits, equipment, or software described in this manual.

Reproduction of the contents of this manual, in whole or in part, without written permission of Rockwell Automation, Inc., is prohibited.

Throughout this manual, when necessary, we use notes to make you aware of safety considerations.

WARNING 	Identifies information about practices or circumstances that can cause an explosion in a hazardous environment, which may lead to personal injury or death, property damage, or economic loss.
IMPORTANT	Identifies information that is critical for successful application and understanding of the product.
ATTENTION 	Identifies information about practices or circumstances that can lead to personal injury or death, property damage, or economic loss. Attentions help you identify a hazard, avoid a hazard, and recognize the consequence
SHOCK HAZARD 	Labels may be on or inside the equipment, for example, a drive or motor, to alert people that dangerous voltage may be present.
BURN HAZARD 	Labels may be on or inside the equipment, for example, a drive or motor, to alert people that surfaces may reach dangerous temperatures.

Allen-Bradley, ControlLogix, Rockwell Automation, RSLogix 5000, ControlFlash, TechConnect, and RSLinx are trademarks of Rockwell Automation, Inc.

Trademarks not belonging to Rockwell Automation are property of their respective companies.

Introduction

This release of this document contains new and updated information.

Updated Information

The document contains the following new chapters:

- Security and RSLogix 5000 Software
- Designing Add-On Instructions
- Using an Add-On Instruction

Notes:

Purpose of this Manual

This manual guides the development of projects for Logix5000 controllers. It provides step-by-step procedures on how to perform the following tasks, which are common to all Logix5000 controllers:

- Organize Tasks, Programs, and Routines
- Organizing Tags
- Designing a Sequential Function Chart
- Program Routines using Ladder logic, Function Block diagram, Sequential Function Chart, or Structured Text programming languages
- Communicate with Other Controllers
- Communicate and Process ASCII Information
- Handle Faults
- Develop and use Add-On Instructions

The term Logix5000 controller refers to any controller that is based on the Logix5000 operating system, such as:

- CompactLogix controllers
- ControlLogix controllers
- DriveLogix controllers
- FlexLogix controllers
- SoftLogix5800 controllers

Who Should Use this Manual

This manual is intended for those individuals who program applications that use Logix5000 controllers, such as:

- software engineers
- control engineers
- application engineers
- instrumentation technicians

When to Use this Manual

Use this manual when you perform these actions:

- develop the basic code for your application
- modify an existing application
- perform isolated tests of your application

As you integrate your application with the I/0 devices, controllers, and networks in your system:

- Refer to the user manual for your specific type of controller.
- Use this manual as a reference, when needed.

How to Use this Manual

As you use this manual, you will see some terms that are formatted differently from the rest of the text:

Text that is	Identifies	For example	Means
Italic	the actual name of an item that you see on your screen or in an example	Right-click *User-Defined* …	Right-click the item that is named User-Defined.
bold	an entry in the "Glossary"	Type a **name** …	If you want additional information, refer to **name** in the "Glossary." If you are viewing the PDF file of the manual, click **name** to jump to the glossary entry.
`courier`	information that you must supply based on your application (a variable)	Right-click *`name_of_program`* …	You must identify the specific program in your application. Typically, it is a name or variable that you have defined.
enclosed in brackets	a keyboard key	Press [Enter].	Press the Enter key.

Important User Information 1-2

Summary of Changes

Introduction . . . . . Summary of Changes-3
Updated Information . . . . . Summary of Changes-3

Preface

Purpose of this Manual 1-5
Who Should Use this Manual 1-5
When to Use this Manual. 1-6
How to Use this Manual 1-6

Table of Contents

Chapter 1

Communicating with I/O

Using This Chapter 1-21
Configure an I/O Module 1-21
- Requested Packet Interval 1-22
- Communication Format 1-23
- Electronic Keying 1-26

Address I/O Data 1-27
Buffering I/O 1-28
- Buffer I/O 1-28

Chapter 2

Organizing Tags

Using this Chapter 2-31
Defining Tags 2-31
- Tag Type. 2-32
- Data Type. 2-33
- Scope 2-35

Guidelines for Tags 2-36
Create a Tag 2-39
Create an Array. 2-39
- Create an Array 2-43

Create a User-Defined Data Type 2-44
- Guidelines for User-Defined Data Types. 2-46
- Create a User-Defined Data Type. 2-46

Describe a User-Defined Data Type 2-48
- Turn Pass-Through and Append Descriptions On or Off 2-49
- Paste a Pass-Through Description 2-49

Address Tag Data. 2-50
Assigning Alias Tags 2-51
- Display Alias Information 2-52
- Assign an Alias. 2-53

Assigning an Indirect Address. 2-54
- Expressions 2-56
- Array Subscript Out of Range. 2-57

Chapter 3

Managing Tasks

Introduction 3-59
Select Controller Tasks 3-60
- Use Caution in the Number of Tasks That You Use. 3-62

Prioritize Periodic and Event Tasks 3-62

Additional Considerations 3-63
Leave Enough Time for Unscheduled Communication 3-65
Avoid Overlaps 3-66
Manually Check for Overlaps 3-67
Programmatically Check for Overlaps 3-68
Configuring Output Processing for a Task 3-70
Manually Configure Output Processing 3-72
Programmatically Configure Output Processing 3-73
Inhibit a Task 3-74
Manually Inhibit or Uninhibit a Task 3-74
Programmatically Inhibit or Uninhibit a Task 3-75
Choose the Trigger for an Event Task 3-77
Using the Module Input Data State Change Trigger 3-79
How an I/O Module Triggers an Event Task 3-79
Make Sure Your Module Can Trigger an Event Task 3-82
Checklist for an Input Event Task 3-83
Estimate Throughput. 3-85
Estimate Throughput. 3-87
Additional Considerations 3-88
Using the Motion Group Trigger 3-88
Checklist for a Motion Group Task 3-89
Using the Axis Registration Trigger 3-90
Checklist for an Axis Registration Task 3-91
Using the Axis Watch Trigger 3-94
Checklist for an Axis Watch Task. 3-95
Using the Consumed Tag Trigger 3-98
Maintain the Integrity of Data. 3-100
Synchronize Multiple Controllers 3-101
Checklist for the Producer Controller 3-102
Checklist for the Consumer Controller 3-103
Producer Controller. 3-104
Consumer Controller. 3-105
Using the EVENT Instruction Trigger. 3-106
Programmatically Determine if an EVENT Instruction Triggered a Task 3-107
Checklist for an EVENT Instruction Task 3-107
Create a Task 3-109
Create an Event Task. 3-109
Create a Periodic Task 3-110
Define a Timeout Value for an Event Task 3-111
Assign a Timeout Value to an Event Task 3-111
Programmatically Configure a Timeout 3-112
Programmatically Determine if a Timeout Occurs 3-113
Adjust the System Overhead Time Slice 3-115
Adjust the System Overhead Time Slice. 3-117
Adjust the Watchdog Time 3-118

Adjust the Watchdog Timer for a Task 3-118

Designing a Sequential Function Chart

Chapter 4

When to Use This Chapter 4-121
What is a Sequential Function Chart? 4-122
Define the Tasks. 4-125
Choose How to Execute the SFC 4-126
Define the Steps of the Process. 4-126
Step Guidelines 4-127
SFC_STEP Structure 4-128
Organize the Steps 4-131
Sequence. 4-133
Selection Branch 4-133
Simultaneous Branch 4-134
Wire to a Previous Step 4-134
Add Actions for Each Step 4-135
How Do You Want to Use the Action? 4-135
Use a Non-Boolean Action 4-136
Use a Boolean Action 4-137
SFC_ACTION Structure 4-138
Describe Each Action in Pseudocode. 4-139
Choose a Qualifier for an Action 4-139
Define the Transition Conditions 4-140
Transition Tag 4-143
How Do You Want to Program the Transition? 4-143
Use a BOOL Expression. 4-143
Call a Subroutine 4-144
Transition After a Specified Time 4-145
Turn Off a Device at the End of a Step 4-148
Choose a Last Scan Option. 4-148
Use the Don't Scan Option. 4-150
Use the Programmatic Reset Option 4-151
Use the Automatic Reset Option 4-153
Keep Something On From Step-to-Step 4-154
How Do You Want to Control the Device? 4-154
Use a Simultaneous Branch. 4-155
Store and Reset an Action 4-155
Use One Large Step. 4-157
End the SFC. 4-158
Use a Stop Element 4-158
Restart (Reset) the SFC 4-159
SFC_STOP Structure 4-160
Nest an SFC 4-161
Pass Parameters 4-162
Configure When to Return to the OS/JSR. 4-162
Pause or Reset an SFC 4-163

Execution Diagrams 4-163

Programming a Sequential Function Chart

Chapter 5

When to Use This Chapter 5-169
Add an SFC Element 5-169
Add and Manually Connect Elements 5-170
Add and Automatically Connect Elements 5-170
Drag and Drop Elements 5-171
Create a Simultaneous Branch 5-171
Start a Simultaneous Branch 5-171
End a Simultaneous Branch 5-172
Create a Selection Branch 5-173
Start a Selection Branch 5-173
End a Selection Branch 5-173
Set the Priorities of a Selection Branch 5-174
Return to a Previous Step 5-175
Connect a Wire to the Step 5-175
Hide a Wire 5-176
Configure a Step 5-176
Assign the Preset Time for a Step 5-176
Configure Alarms for a Step 5-177
Use an Expression to Calculate a Time 5-177
Program a Transition 5-178
Enter a BOOL Expression 5-178
Call a Subroutine 5-179
Add an Action 5-180
Configure an Action 5-180
Change the Qualifier of an Action 5-180
Calculate a Preset Time at Runtime 5-181
Mark an Action as a Boolean Action 5-182
Program an Action 5-182
Enter Structured Text 5-182
Call a Subroutine 5-183
Assign the Execution Order of Actions 5-184
Document an SFC 5-184
Add Structured Text Comments 5-185
Add a Tag Description 5-186
Add a Text Box 5-186
Show or Hide Text Boxes or Tag Descriptions 5-187
Hide an Individual Tag Description 5-188
Configure the Execution of the SFC 5-188
Verify the Routine 5-189
Edit an SFC Online 5-190

Chapter 6

Programming Structured Text

When to Use This Chapter 6-191
Structured Text Syntax. 6-191
Assignments 6-192
Specify a non-retentive assignment. 6-193
Assign an ASCII character to a string. 6-194
Expressions. 6-194
Use arithmetic operators and functions 6-196
Use relational operators. 6-197
Use logical operators 6-199
Use bitwise operators. 6-200
Determine the order of execution. 6-200
Instructions. 6-201
Constructs. 6-202
Some key words are reserved for future use. 6-202
IF...THEN 6-203
CASE...OF 6-206
FOR…DO 6-209
WHILE…DO 6-212
REPEAT…UNTIL 6-215
Comments 6-218

Chapter 7

Programming Ladder Diagram

When to Use This Chapter 7-219
Definitions 7-219
Instruction 7-219
Branch 7-220
Rung Condition 7-221
Write Ladder Logic 7-222
Choose the Required Instructions 7-222
Arrange the Input Instructions 7-223
Arrange the Output Instructions 7-224
Choose a Tag Name for an Operand 7-224
Enter Ladder Logic 7-225
Append an Element to the Cursor Location 7-226
Drag and Drop an Element. 7-226
Assign Instruction Operands. 7-227
Create and Assign a New Tag 7-227
Choose a Name or an Existing Tag 7-228
Drag a Tag From the Tags Window. 7-228
Assign an Immediate (Constant) Value 7-228
Verify the Routine 7-229

Chapter 8

Programming a Function Block Diagram

When to Use This Chapter 8-231
Identify the Sheets for the Routine 8-231

Choose the Function Block Elements 8-232
Choose a Tag Name for an Element. 8-233
Define the Order of Execution 8-235
Data Latching. 8-235
Order of Execution 8-237
Resolve a Loop. 8-238
Resolve Data Flow Between Two Blocks. 8-240
Create a One Scan Delay 8-240
Summary. 8-241
Identify any Connectors. 8-241
Define Program/Operator Control. 8-242
Add a Sheet. 8-245
Add a Function Block Element 8-245
Connect Elements 8-246
Show or Hide a Pin 8-246
Wire Elements Together 8-247
Mark a Wire with the Assume Data Available Indicator 8-247
Assign a Tag 8-248
Create and Assign a New Tag 8-248
Assign an Existing Tag 8-249
Assign an Immediate Value (Constant). 8-249
Use an IREF. 8-249
Enter a Value in the Tag of a Block 8-250
Connect Blocks with an OCON and ICON 8-250
Add an OCON 8-250
Add an ICON 8-251
Verify the Routine 8-251

Chapter 9

Designing Add-On Instructions

Introduction 9-253
What is an Add-On Instruction? 9-254
What are the Parts of an Add-On Instruction Definition? . . . 9-255
Deciding to Develop an Add-On Instruction. 9-259
Instruction Functionality 9-259
Encapsulation. 9-259
Available Languages. 9-259
Instruction Size 9-260
Runtime Editing of Functionality 9-260
Nesting Add-On Instructions 9-260
Using a Routine Versus an Add-On Instruction 9-261
Programmatic Access to Data. 9-262
Unavailable Instructions within Add-On Instructions. 9-263
Planning the Design of an Add-On Instruction 9-265
Developing an Add-On Instruction 9-265
Creating an Add-On Instruction 9-267

Considerations when Selecting a Language for an Add-On Instruction 9-268
Create Parameters 9-270
Considerations When Passing Arguments to Parameters by Reference or by Value 9-273
Considerations for Selecting a Data Type for a Parameter . . . 9-273
Consideration when Using Single Dimension Array as InOut Parameter 9-273
Determining Which Parameters to Make Visible or Required 9-274
Create Local Tags. 9-276
Enter the Logic of the Add-On Instruction 9-279
Add-On Instructions Execution Considerations 9-279
Optimizing Performance Considerations 9-279
Defining the Operation of an Instruction in Different Scan Modes. . . . 9-280
Enabling Scan Modes. 9-281
Prescan Routine 9-281
Postscan Routine 9-283
EnableInFalse Routine 9-285
Using the EnableIn and EnableOut Parameters. 9-287
EnableIn Parameter and Ladder Diagram 9-287
EnableIn Parameter and Function Blocks 9-287
EnableIn Parameter and Structured Text 9-288
Testing the Add-On Instruction 9-289
Before You Test. 9-289
Test the Flow 9-289
Monitor Logic with Data Context Views 9-289
Tips for Validating the Add-On Instruction in Different Scan Modes 9-290
Managing Add-On Instruction Definitions 9-292
Storing Your Instructions 9-292
Updating an Add-On Instruction to a Newer Revision. 9-294
Defining Source Protection for an Add-On Instruction 9-298
Enable the Source Protection Feature 9-299
Apply Source Protection 9-300
Observe Source Protection 9-302
Creating Instruction Help 9-303
Define the Documentation for Add-On Instructions 9-303
Write Clear Descriptions 9-303
Document an Add-On Instruction 9-305
Motor Starter Instruction Example 9-307
Motor Starter Ladder Diagram 9-309
Motor Starter Function Block Diagram 9-309
Motor Starter Structured Text. 9-309
Simulation Instruction Example 9-310
Ladder Diagram Configuration 9-311

Function Block Diagram Configuration 9-312
Structured Text Configuration 9-312

Chapter 10

Using an Add-On Instruction

Introduction 10-313
What is an Add-On Instruction? 10-313
Adding an Add-On Instruction to Your Routine 10-314
Programming Tips when Using an Add-On Instruction 10-319
Programmatically Accessing a Parameter 10-319
Using the Jog Command in Ladder Diagram 10-320
Using the Jog Command In Function Block Diagram 10-320
Using the Jog Command in Structured Text 10-322
Monitoring the Value of a Parameter 10-323
View Logic and Monitor with Data Context 10-324
Is the Add-On Instruction Source Protected? 10-326
Copying an Add-On Instruction Definition 10-327
Importing an Add-On Instruction Definition 10-328
Exporting an Add-On Instruction Definition 10-329

Chapter 11

Communicating with Other Devices

When to Use This Chapter 11-331
Connections 11-331
Inhibit a Connection 11-332
Manage a Connection Failure 11-334
Produce and Consume a Tag 11-339
Controllers and Networks that Support Produced/Consumed Tags 11-340
Connection Requirements of a Produced or Consumed Tag 11-340
Organize Tags for Produced or Consumed Data 11-341
Adjust for Bandwidth Limitations 11-342
Produce a Tag 11-343
Consume Data That Is Produced by Another Controller . . . 11-344
Additional Steps for a PLC-5C Controller 11-346
Execute a Message (MSG) Instruction 11-347
Message Queue 11-349
Cache List 11-350
Unconnected Buffers 11-351
Guidelines 11-352
Get or Set the Number of Unconnected Buffers 11-353
Get the Number of Unconnected Buffers 11-353
Set the Number of Unconnected Buffers 11-354
Convert Between INTs and DINTs 11-356

Chapter 12

Producing a Large Array

When to Use This Chapter 12-359

Produce a Large Array 12-360

Chapter 13

Communicating with an ASCII Device

When to Use This Chapter 13-365
Connect the ASCII Device 13-366
Configure the Serial Port 13-367
Configure the User Protocol 13-368
Create String Data Types 13-370
Read Characters from the Device 13-371
Send Characters to the Device. 13-373
Enter ASCII Characters 13-375

Chapter 14

Processing ASCII Characters

When to Use This Chapter 14-377
Extract a Part of a Bar Code 14-378
Look Up a Bar Code 14-379
Create the PRODUCT_INFO Data Type 14-380
Search for the Characters 14-380
Identify the Lane Number. 14-381
Reject Bad Characters 14-381
Enter the Product IDs and Lane Numbers 14-382
Check the Bar Code Characters. 14-382
Convert a Value 14-383
Decode an ASCII Message 14-384
Build a String 14-385

Chapter 15

Forcing Logic Elements

When to Use This Chapter 15-387
Precautions 15-388
Enable Forces 15-388
Disable or Remove a Force 15-389
Check Force Status. 15-389
FORCE LED. 15-390
GSV Instruction. 15-390
What to Force. 15-391
When to Use an I/O Force 15-391
Force an Input Value 15-392
Force an Output Value 15-392
Add an I/O Force 15-392
When to Use Step Through 15-393
Step Through a Transition or a Force of a Path. 15-393
When to Use an SFC Force 15-393
Force a Transition 15-394
Force a Simultaneous Path 15-395
Add an SFC Force 15-395

Remove or Disable Forces. 15-396
Remove an Individual Force 15-397
Disable All I/O Forces 15-397
Remove All I/O Forces 15-397
Disable All SFC Forces 15-398
Remove All SFC Forces 15-398

Chapter 16

Handle a Major Fault

Using This Chapter 16-399
Develop a Fault Routine 16-399
Choose Where to Place the Fault Routine 16-400
Create a Fault Routine for a Program. 16-400
Create a Routine for the Controller Fault Handler 16-401
Create a Routine for the Power-Up Handler 16-402
Programmatically Clear a Major Fault. 16-403
Create a Data Type to Store Fault Information 16-403
Get the Fault Type and Code 16-404
Check for a Specific Fault 16-404
Clear the Fault 16-405
Clear a Major Fault During Prescan 16-405
Identify When the Controller is in Prescan 16-406
Get the Fault Type and Code 16-406
Check for a Specific Fault 16-407
Clear the Fault 16-408
Test a Fault Routine. 16-408
Create a User-Defined Major Fault. 16-409
Create a Fault Routine for the Program 16-410
Configure the Program to Use the Fault Routine 16-410
Jump to the Fault Routine. 16-410
Major Fault Codes 16-412

Chapter 17

Monitor Minor Faults

When to Use This Chapter 17-415
Monitor Minor Faults. 17-415
Minor Fault Codes 17-418

Chapter 14

Access Status Information

When to Use This Chapter 18-421
Monitor Status Flags 18-421
The Status of S:FS When the Project Has an SFC. 18-422
Get and Set System Data 18-422

Chapter 15

Security and RSLogix 5000 Software

Introduction 19-425
Understanding the Current Relationship Between FactoryTalk Security and RSI Security Server 19-425
About Migrating RSI Security Server to FactoryTalk Security. . . 19-426
How FactoryTalk Security Differs from RSI Security Server 19-426
Differences between RSI Security Server and FactoryTalk Security 19-427
Support for resource groups and action groups. 19-427
Unsupported features 19-427
Setting Up Source Protection 19-428
Source Protect Components 19-429
Using Routine Source Protection 19-431
Choose the Level of Protection for Each Routine. 19-434
Choose the Number of Source Keys 19-434
Define the Source Key or Keys. 19-435
Choose a File Location in Which to Store the Source Keys . 19-435
Create a File for the Source Keys 19-435
Protect a Routine with a Source Key 19-436
View Protected Routine. 19-437
Remove Access to a Protected Routine 19-438
Disable Routine Source Protection. 19-439
Gain Access to a Protected Routine 19-440
Using FactoryTalk Security to Protect a Project. 19-442
Migrating to FactoryTalk Security 19-442
Using the Security Server 19-443

Chapter 16

Store and Load a Project Using Nonvolatile Memory

When to Use This Chapter 20-445
Before You Use Nonvolatile Memory 20-446
Choose a Controller That Has Nonvolatile Memory. 20-446
Prevent a Major Fault During a Load. 20-447
Format a CompactFlash Card 20-447
Determine How to Handle Firmware Updates 20-448
Choose When to Load an Image 20-449
Examples 20-450
Store a Project 20-451
Load a Project. 20-453
Check for a Load 20-455
Clear Nonvolatile Memory 20-456
Change the Load Image Option 20-456
Clear the Project from the Controller. 20-457
Store the Empty Image 20-457
Use a CompactFlash Reader 20-458
Manually Change Which Project Loads from the CompactFlash Card 20-458
Manually Change the Load Parameters for a Project. 20-460

Other Uses for a CompactFlash Card 20-461

Chapter 17

Determine Controller Memory Information

When to Use This Chapter 21-463
Determine What Memory Information You Want 21-463
Estimate Memory Information Offline. 21-464
View Run Time Memory Information 21-465
Write Logic to Get Memory Information. 21-466
Get Memory Information from the Controller 21-466
Choose the Memory Information That You Want 21-467
Convert INTs to a DINT 21-468

Appendix A

Manage Multiple Messages

When to Use This Appendix. A-469
How to Use this Appendix A-469
Message Manager Logic. A-470
Initialize the Logic A-470
Restart the Sequence, If Required. A-470
Send the First Group of MSGs. A-470
Enable the Next Group of MSGs. A-471
Send the Next Group of MSGs A-471
Enable the Next Group of MSGs. A-472
Send the Next Group of MSGs A-472

Appendix B

Send a Message to Multiple Controllers

When to Use This Appendix. B-473
Set Up the I/O Configuration B-474
Define Your Source and Destination Elements. B-475
Create the MESSAGE_CONFIGURATION Data Type . . . B-476
Create the Configuration Array. B-477
Get the Size of the Local Array. B-479
Load the Message Properties for a Controller B-480
Configure the Message B-481
Step to the Next Controller. B-482
Restart the Sequence B-482

Appendix C

IEC61131-3 Compliance

Use This Appendix. C-483
Introduction C-483
Operating System C-484
Data Definitions. C-484
Programming Languages C-485
Instruction Set C-486
IEC61131-3 Program Portability. C-486
IEC Compliance Tables. C-487

Glossary

A Glossary-495
B Glossary-498
C Glossary-500
D Glossary-503
E Glossary-504
F Glossary-505
H Glossary-506
I Glossary-506
L Glossary-507
M Glossary-508
N Glossary-512
O Glossary-512
P Glossary-515
Q Glossary-519
R Glossary-520
S Glossary-521
T Glossary-529
U Glossary-530
W Glossary-532

Index

ASCII Character Codes 1-545
Rockwell Automation Support 1-547
Installation Assistance 1-547
New Product Satisfaction Return 1-547

Communicating with I/O

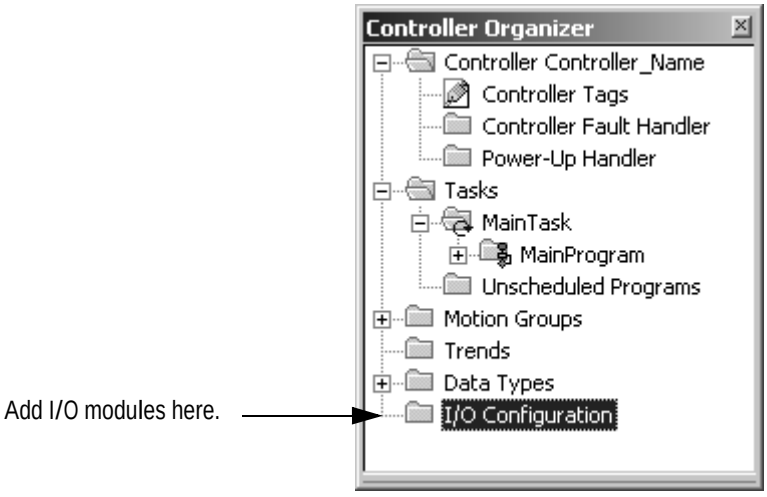
Using This Chapter

This chapter provides basic information on how a Logix5000 controller communicates with I/O modules.

Topic	Page
Configure an I/O Module	21
Address I/O Data	27
Buffering I/O	28

Configure an I/O Module

To communicate with an I/O module in your system, you add the module to the I/O Configuration folder of the controller.



When you add the module, you also define a specific configuration for the module. While the configuration options vary from module to module, there are some common options that you typically configure:

- Requested Packet Interval
- Communication Format
- Electronic Keying

Requested Packet Interval

The Logix5000 controller uses connections to transmit I/O data.

Term	Definition
Connection	A communication link between two devices, such as between a controller and an I/O module, PanelView terminal, or another controller. Connections are allocations of resources that provide more reliable communications between devices than unconnected messages. The number of connections that a single controller can have is limited. You indirectly determine the number of connections the controller uses by configuring the controller to communicate with other devices in the system. The following types of communication use connections: • I/O modules • produced and consumed tags • certain types of Message (MSG) instructions (not all types use a connection)
requested packet interval (RPI)	The RPI specifies the period at which data updates over a connection. For example, an input module sends data to a controller at the RPI that you assign to the module. • Typically, you configure an RPI in milliseconds (ms). The range is 0.2 ms (200 microseconds) to 750 ms. • If a ControlNet network connects the devices, the RPI reserves a slot in the stream of data flowing across the ControlNet network. The timing of this slot may not coincide with the exact value of the RPI, but the control system guarantees that the data transfers at least as often as the RPI.

In Logix5000 controllers, I/O values update at a period that you configure via the I/O configuration folder of the project. The values update asynchronous to the execution of logic. At the specified interval, the controller updates a value independently from the execution of logic.

ATTENTION



Take care to ensure that data memory contains the appropriate values throughout a task's execution. You can duplicate or buffer data at the beginning of the scan to provide reference values for your logic.

- Programs within a task access input and output data directly from controller-scoped memory.
- Logic within any task can modify controller-scoped data.
- Data and I/O values are asynchronous and can change during the course of a task's execution.
- An input value referenced at the beginning of a task's execution can be different when referenced later.
- To prevent an input value from changing during a scan, copy the value to another tag and use the data from there (buffer the values). To buffer your I/O values, see page 28.

Communication Format

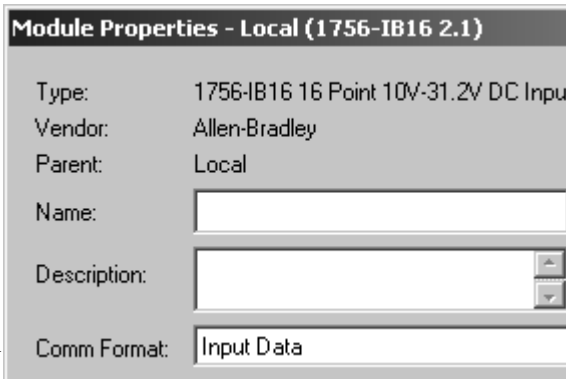
The communication format that you choose determines the data structure for the tags that are associated with the module. Many I/O modules support different formats. Each format uses a different data structure. The communication format that you choose also determines:

- Direct or Rack-Optimized Connection
- Ownership

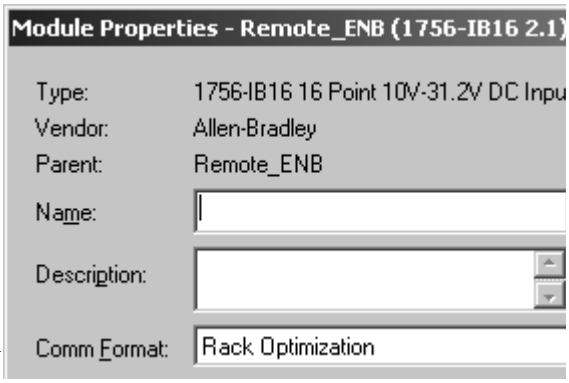
Direct or Rack-Optimized Connection

The Logix5000 controller uses connections to transmit I/O data. These connections can be direct connections or rack-optimized connections.

Term	Definition
direct connection	A direct connection is a real-time, data transfer link between the controller and an I/O module. The controller maintains and monitors the connection with the I/O module. Any break in the connection, such as a module fault or the removal of a module while under power, sets fault bits in the data area associated with the module.
rack-optimized connection	For digital I/O modules, you can select rack-optimized communication. A rack-optimized connection consolidates connection usage between the controller and all the digital I/O modules in the chassis (or DIN rail). Rather than having individual, direct connections for each I/O module, there is one connection for the entire chassis (or DIN rail).



A direct connection is any connection that *does not* use the Rack Optimization Come Format.



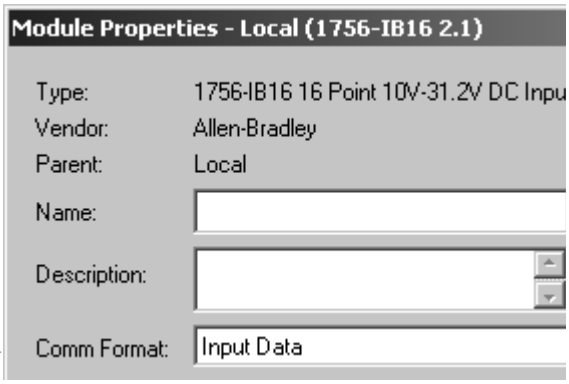
rack-optimized connection

Ownership

In a Logix5000 system, modules multicast data. This means that multiple devices can receive the same data at the same time from a single device.

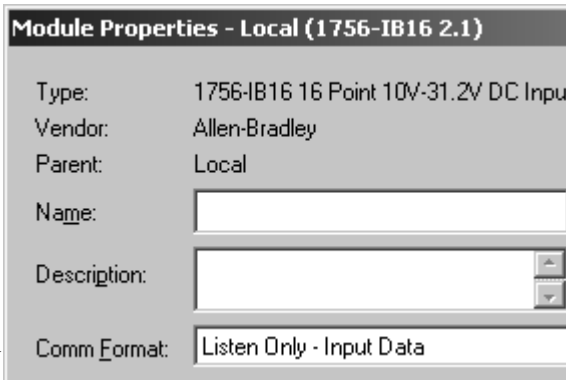
When you choose a communication format, you have to choose whether to establish an owner or listen-only relationship with the module.

owner controller The controller that creates the primary configuration and communication connection to a module. The owner controller writes configuration data and can establish a connection to the module.



An owner connection is any connection that does not include Listen-Only in its Comm Format.

listen-only connection An I/O connection where another controller owns/provides the configuration data for the I/O module. A controller using a listen-only connection only monitors the module. It does not write configuration data and can only maintain a connection to the I/O module when the owner controller is actively controlling the I/O module.



listen-only connection

Use the following table to choose the type of ownership for a module:

If the module is an	And another controller	And you want to	Then use this type of connection
input module	does not own the module	→	owner (i.e., not listen-only)
	owns the module	maintain communication with the module if it loses communication with the other controller	owner (i.e., not listen-only) Use the same configuration as the other owner controller.
		stop communication with the module if it loses communication with the other controller	listen-only
output module	does not own the module	→	owner (i.e., not listen-only)
	owns the module	→	listen-only

There is a noted difference in controlling input modules versus controlling output modules.

Controlling	This ownership	Description
input modules	owner	An input module is configured by a controller that establishes a connection as an owner. This configuring controller is the first controller to establish an owner connection. Once an input module has been configured (and owned by a controller), other controllers can establish owner connections to that module. This lets additional owners to continue to receive multicast data if the original owner controller breaks its connection to the module. All other additional owners must have the identical configuration data and identical communications format that the original owner controller has, otherwise the connection attempt is rejected.
	listen-only	Once an input module has been configured (and owned by a controller), other controllers can establish a listen-only connection to that module. These controllers can receive multicast data while another controller owns the module. If all owner controllers break their connections to the input module, all controllers with listen-only connections no longer receive multicast data.
output modules	owner	An output module is configured by a controller that establishes a connection as an owner. Only one owner connection is allowed for an output module. If another controller attempts to establish an owner connection, the connection attempt is rejected.
	listen-only	Once an output module has been configured (and owned by one controller), other controllers can establish listen-only connections to that module. These controllers can receive multicast data while another controller owns the module. If the owner controller breaks its connection to the output module, all controllers with listen-only connections no longer receive multicast data.

Electronic Keying

ATTENTION



Be careful when you disable electronic keying. If used incorrectly, this option can lead to personal injury or death, property damage, or economic loss.

When you configure a module, you specify the slot number for the module. However, it is possible to place a different module in that slot, either on purpose or accidently.

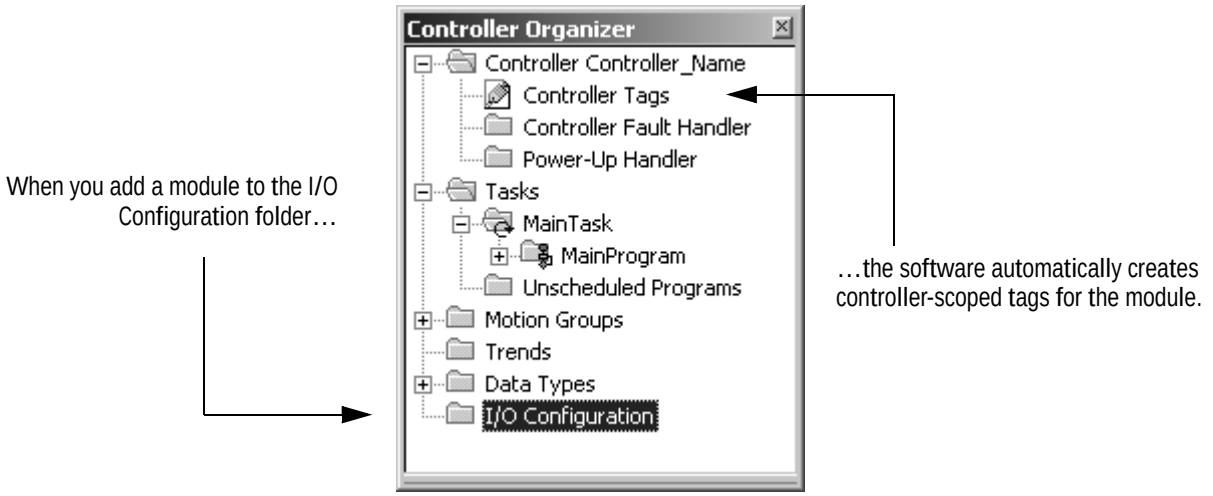
Electronic keying lets you protect your system against the accidental placement of the wrong module in a slot. The keying option you choose determines how closely any module in a slot must match the configuration for that slot.

If	Then Select
all information must match: • type • catalog number • vendor • major and minor revision number	Exact Match
all information except the minor revision number	Compatible Module
no information must match	Disable Keying

Address I/O Data

I/O information is presented as a set of tags.

- Each tag uses a structure of data. The structure depends on the specific features of the I/O module.
- The name of the tags is based on the location of the I/O module in the system.



An I/O address follows this format:

`Location` `:Slot` `:Type` `.Member` `.SubMember` `.Bit`

= Optional

Where	Is
`Location`	Network location LOCAL = same chassis or DIN rail as the controller *ADAPTER_NAME* = identifies remote communication adapter or bridge module
`Slot`	Slot number of I/O module in its chassis or DIN rail
`Type`	Type of data I = input O = output C = configuration S = status
`Member`	Specific data from the I/O module; depends on what type of data the module can store. • For a digital module, a Data member usually stores the input or output bit values. • For an analog module, a Channel member (CH#) usually stores the data for a channel.
`SubMember`	Specific data related to a Member.
`Bit`	Specific point on a digital I/O module; depends on the size of the I/O module (0-31 for a 32-point module)

Buffering I/O

Buffering is a technique in which logic does not directly reference or manipulate the tags of real I/O devices. Instead, the logic uses a copy of the I/O data. Buffer I/O in the following situations:

- To prevent an input or output value from changing during the execution of a program. (I/O updates **asynchronous** to the execution of logic.)
- To copy an input or output tag to a member of a structure or element of an array.

Buffer I/O

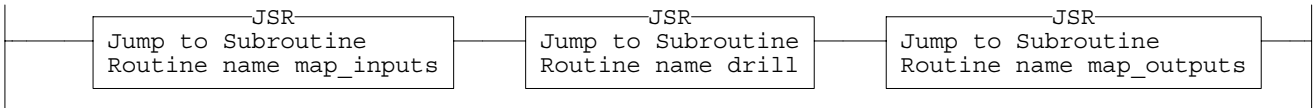
To buffer I/O, perform these actions:

1. On the rung before the logic for the function (s), copy or move the data from the required input tags to their corresponding buffer tags.
2. In the logic of the function (s), reference the buffer tags.
3. On the rung after the function (s), copy the data from the buffer tags to the corresponding output tags.

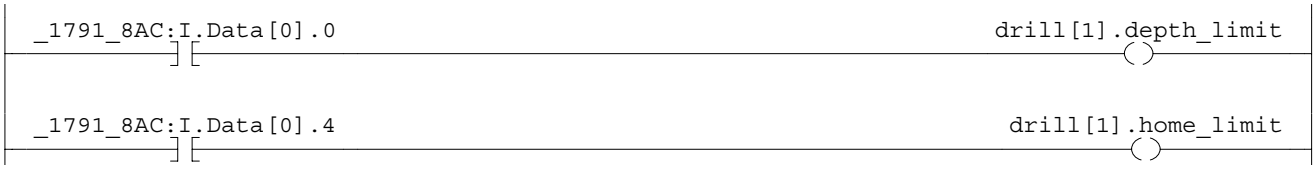
The following example copies inputs and outputs to the tags of a structure for a drill machine.

EXAMPLE Buffer I/O

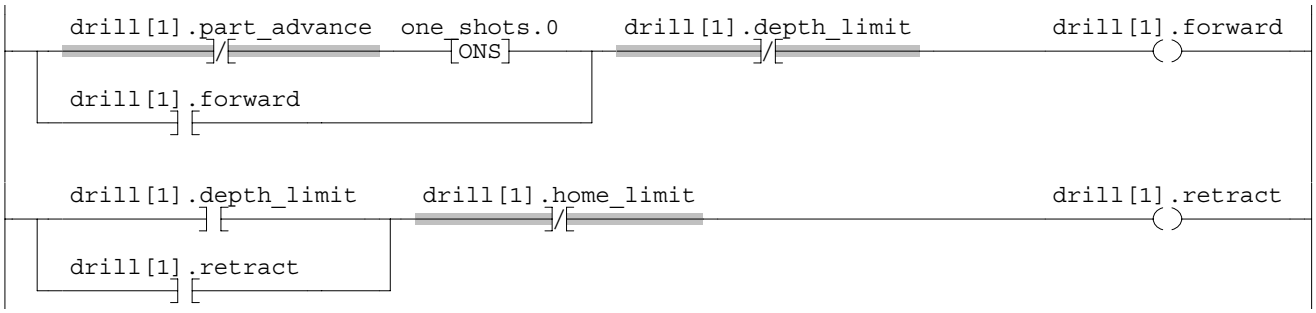
The main routine of the program executes the following subroutines in this sequence.



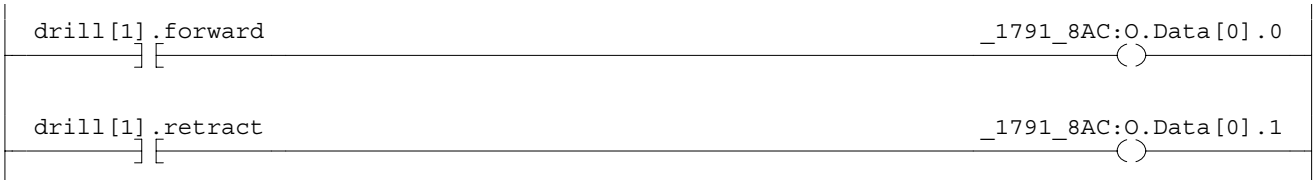
The map_inputs routine copies the values of input devices to their corresponding tags that are used in the drill routine.



The drill routine executes the logic for the drill machine.



The map_outputs routine copies the values of output tags in the drill routine to their corresponding output devices.



42369

The following example uses the CPS instruction to copy an array of data that represent the input devices of a DeviceNet network.

EXAMPLE Buffer I/O

Local:0:I.Data stores the input data for the DeviceNet network that is connected to the 1756-DNB module in slot 0. To synchronize the inputs with the application, the CPS instruction copies the input data to input_buffer.

- While the CPS instruction copies the data, no I/O updates can change the data.
- As the application executes, it uses for its inputs the input data in input_buffer.



Organizing Tags

Using this Chapter

Use this chapter to organize the data for your Logix5000 controller.

Topic	Page
Defining Tags	31
Guidelines for Tags	36
Create a Tag	39
Create an Array	39
Create a User-Defined Data Type	44
Describe a User-Defined Data Type	48
Address Tag Data	50
Assigning Alias Tags	51
Assigning an Indirect Address	54

Defining Tags

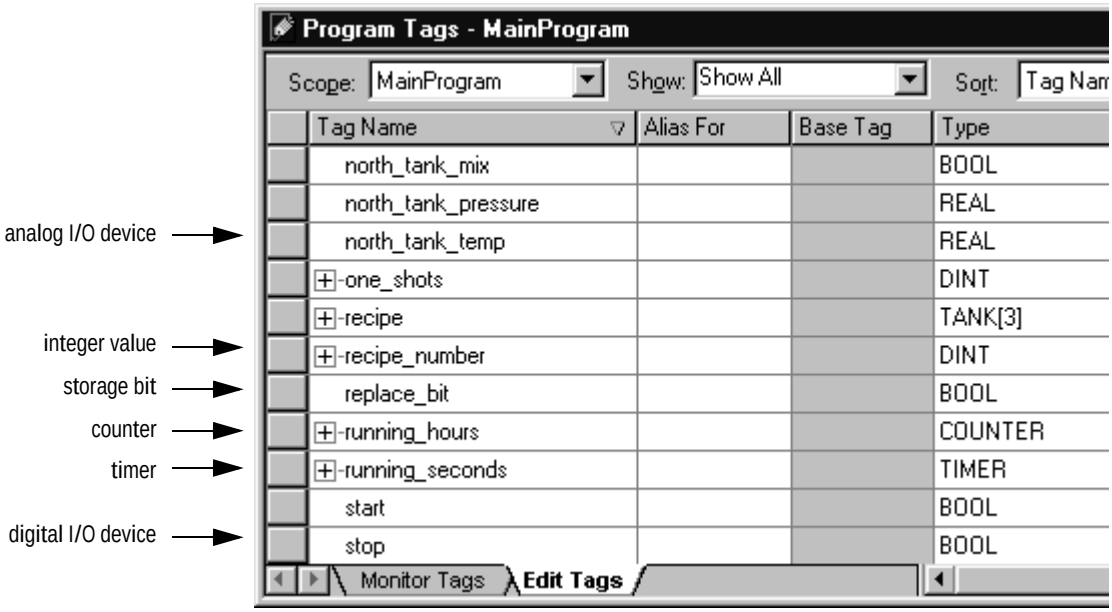
With a Logix5000 controller, you use a tag (alphanumeric name) to address data (variables).

Term	Definition
tag	A text-based name for an area of the controller's memory where data is stored. • Tags are the basic mechanism for allocating memory, referencing data from logic, and monitoring data. • The minimum memory allocation for a tag is four bytes. • When you create a tag that stores data that requires less than four bytes, the controller allocates four bytes, but the data only fills the part it needs.

The controller uses the tag name internally and doesn't need to cross-reference a physical address.

- In conventional programmable controllers, a physical address identifies each item of data.
  - Addresses follow a fixed, numeric format that depend on the type of data, such as N7:8, F8:3.
  - Symbols are required to make logic easier to interpret.
- In Logix5000 controllers, there is no fixed, numeric format. The tag name itself identifies the data. This lets you:
  - organize your data to mirror your machinery
  - document (through tag names) your application as you develop it

EXAMPLE Tags



Tag Type

The tag type defines how the tag operates within your project.

If you want the tag to	Then choose this type
store a value or values for use by logic within the project	Base
represent another tag.	Alias
send data to another controller	Produced
receive data from another controller	Consumed

If you plan to use produced or consumed tags, you must follow additional guidelines as you organize your tags.

Refer to "Communicating with Other Devices" on page 331.

Data Type

Term	Definition
data type	The data type defines the type of data that a tag stores, such as a bit, integer, floating-point value, string, etc.
structure	A data type that is a combination of other data types. • A structure is formatted to create a unique data type that matches a specific need. • Within a structure, each individual data type is called a member. • Like tags, members have a name and data type. • A Logix5000 controller contains a set of predefined structures (data types) for use with specific instructions such as timers, counters, Function Blocks, etc. • You can create your own structures, called a user-defined data type.

The following table outlines the most common data types and when to use each.

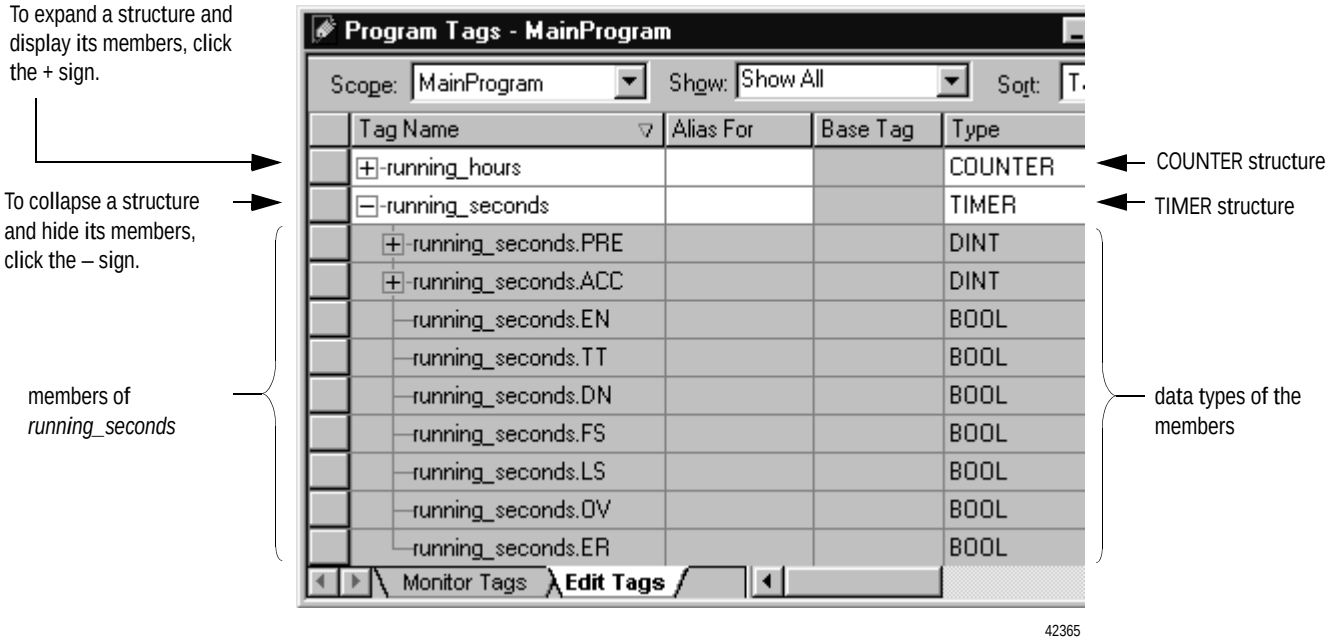
Table 2.1 Data Types

For	Select
analog device in floating-point mode	REAL
analog device in integer mode (for very fast sample rates)	INT
ASCII characters	string
bit	BOOL
counter	COUNTER
digital I/O point	BOOL
floating-point number	REAL
integer (whole number)	DINT
sequencer	CONTROL
timer	TIMER

The minimum memory allocation for a tag is 4 bytes. When you create a tag that stores data that requires less than four bytes, the controller allocates 4 bytes, but the data only fills the part it needs.

Data type	Bits			
	31 16	15 8	7 1	0
BOOL	not used			0 or 1
SINT	not used		-128 to +127	
INT	not used	-32,768 to +32767		
DINT	-2,147,483,648 to +2,147,483,647			
REAL	$-3.40282347E^{38}$ to $-1.17549435E^{-38}$ (negative values) 0 $1.17549435E^{-38}$ to $3.40282347E^{38}$ (positive values)			

The COUNTER and TIMER data types are examples of commonly used structures.

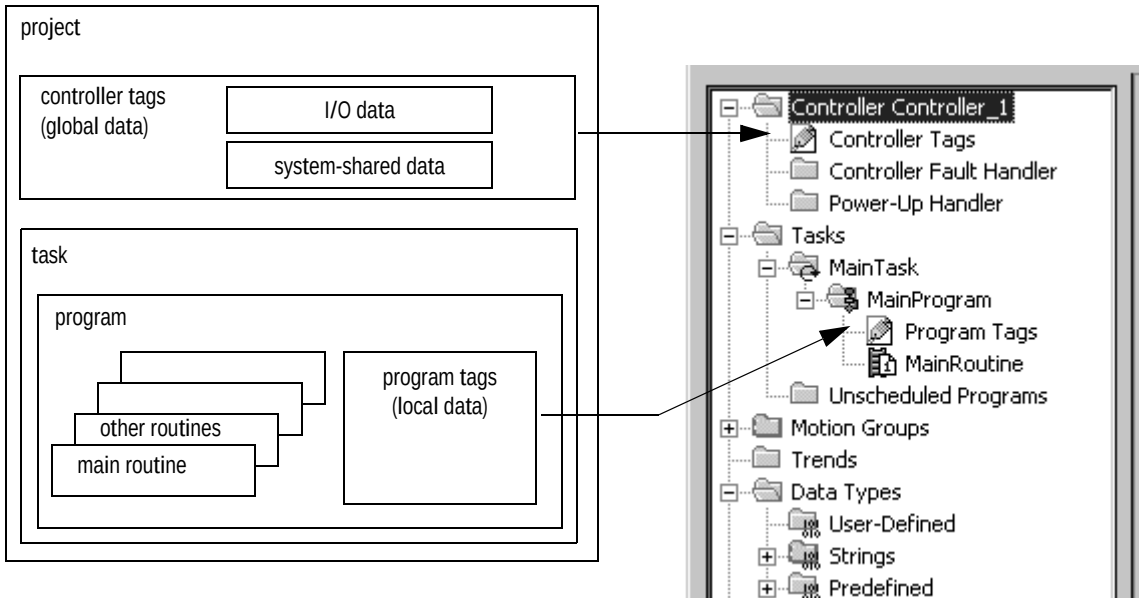


To copy data to a structure, use the COP instruction.

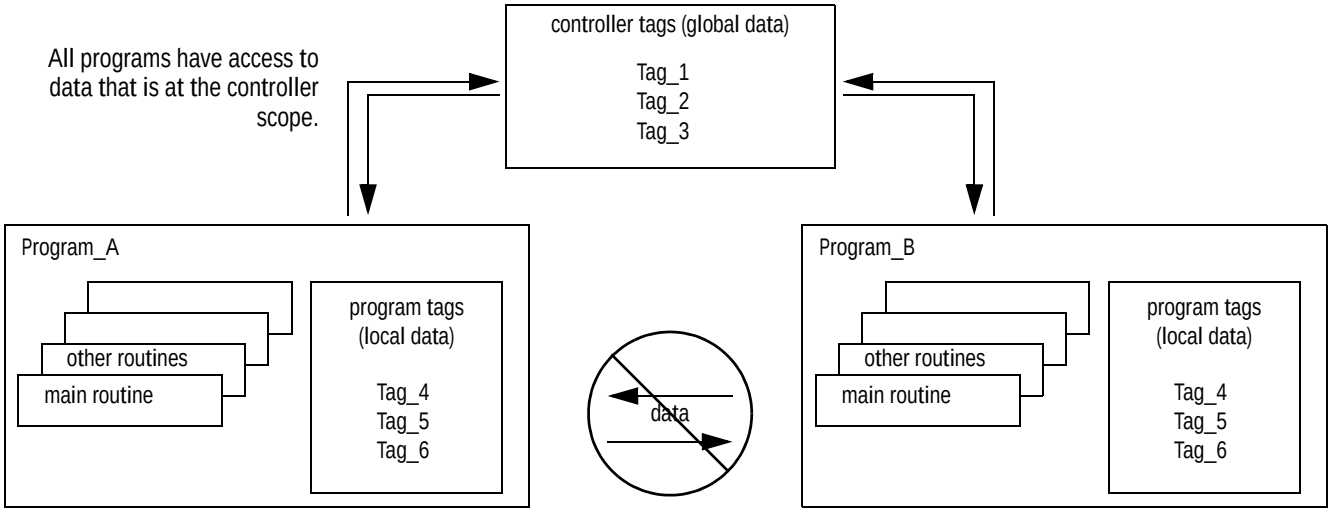
See the *Logix5000 Controllers General Instruction Set Reference Manual*, publication 1756-RM003.

Scope

When you create a tag, you define it as either a controller tag (global data) or a program tag for a specific program (local data).



A Logix5000 controller lets you divide your application into multiple programs, each with its own data. There is no need to manage conflicting tag names between programs. This makes it easier to re-use both code and tag names in multiple programs.



Data at the program scope is isolated from other programs:

- Routines cannot access data that is at the program scope of another program.
- You can re-use the tag name of a program-scoped tag in multiple programs.

  For example, both Program_A and Program_B can have a program tag named Tag_4.

Avoid using the same name for a both controller tag and a program tag. Within a program, you cannot reference a controller tag if a tag of the same name exists as a program tag for that program.

Certain tags must be controller scope (controller tag).

If you want to use the tag	Then assign this scope
in more than one program in the project	controller scope (controller tags)
in a Message (MSG) instruction	
to produce or consume data	
to communicate with a PanelView terminal	
none of the above	program scope (program tags)

Guidelines for Tags

Use the following guidelines to create tags for a Logix5000 project:

Guideline	Details
❑ 1. Create user-defined data types.	User-defined data types (structures) let you organize data to match your machine or process. A user-defined data type provides these advantages: • One tag contains all the data related to a specific aspect of your system. This keeps related data together and easy to locate, regardless of its data type. • Each individual piece of data (member) gets a descriptive name. This automatically creates an initial level of documentation for your logic. • You can use the data type to create multiple tags with the same data lay-out. For example, use a user-defined data type to store all the parameters for a tank, including temperatures, pressures, valve positions, and preset values. Then create a tag for each of your tanks based on that data type.
❑ 2. Use arrays to quickly create a group of similar tags.	An array creates multiple instances of a data type under a common tag name. • Arrays let you organize a block of tags that use the same data type and perform a similar function. • You organize the data in 1, 2, or 3 dimensions to match what the data represents. For example, use a 2 dimension array to organize the data for a tank farm. Each element of the array represents a single tank. The location of the element within the array represents the geographic location of the tank. Important: Minimize the use of BOOL arrays. Many array instructions do not operate on BOOL arrays. This makes it more difficult to initialize and clear an array of BOOL data. • Typically, use a BOOL array for the bit-level objects of a PanelView screen. • Otherwise, use the individual bits of a DINT tag or an array of DINTs.

<table>
<tr><th colspan="2">Guideline</th><th colspan="2">Details</th></tr>
<tr><td>❑</td><td>3. Take advantage of program-scoped tags.</td><td colspan="2">If you want multiple tags with the same name, define each tag at the program scope (program tags) for a different program. This lets you re-use both logic and tag names in multiple programs.

Avoid using the same name for both a controller tag and a program tag. Within a program, you cannot reference a controller tag if a tag of the same name exists as a program tag for that program.

Certain tags must be controller scope (controller tag).</td></tr>
<tr><td></td><td></td><td>If you want to use the tag</td><td>Then assign this scope</td></tr>
<tr><td></td><td></td><td>in more than one program in the project</td><td rowspan="4">controller scope (controller tags)</td></tr>
<tr><td></td><td></td><td>in a Message (MSG) instruction</td></tr>
<tr><td></td><td></td><td>to produce or consume data</td></tr>
<tr><td></td><td></td><td>to communicate with a PanelView terminal</td></tr>
<tr><td></td><td></td><td>none of the above</td><td>program scope (program tags)</td></tr>
<tr><td>❑</td><td>4. For integers, use the DINT data type.</td><td colspan="2">To increase the efficiency of your logic, minimize the use of SINT or INT data types. Whenever possible, use the DINT data type for integers.
• A Logix5000 controller typically compares or manipulates values as 32-bit values (DINTs or REALs).
• The controller typically converts a SINT or INT value to a DINT or REAL value before it uses the value.
• If the destination is a SINT or INT tag, the controller typically converts the value back to a SINT or INT value.
• The conversion to or from SINTs or INTs occurs automatically with no extra programming. But it takes extra execution time and memory.</td></tr>
</table>

<table>
<tr><th colspan="2">Guideline</th><th>Details</th></tr>
<tr><td>❑</td><td>5. Limit a tag name to 40 characters.</td><td>Here are the rules for a tag name:
only alphabetic characters (A-Z or a-z), numeric characters (0-9), and underscores (_)
must start with an alphabetic character or an underscore
no more than 40 characters
no consecutive or trailing underscore characters (_)
not case sensitive</td></tr>
<tr><td>❑</td><td>6. Use mixed case.</td><td>Although tags are not case sensitive (upper case A is the same as lower case a), mixed case is easier to read.<table><tr><th>These tags are easier to read</th><th>Than these tags</th></tr><tr><td>Tank_1</td><td>TANK_1</td></tr><tr><td>Tank1</td><td>TANK1</td></tr><tr><td></td><td>tank_1</td></tr><tr><td></td><td>tank1</td></tr></table></td></tr>
<tr><td>❑</td><td>7. Consider the alphabetical order of tags.</td><td>RSLogix 5000 software displays tags of the same scope in alphabetical order. To make it easier to monitor related tags, use similar starting characters for tags that you want to keep together.
Starting each tag for a tank with Tank keeps the tags together.<table><tr><th>Tag Name</th></tr><tr><td>Tank_North</td></tr><tr><td>Tank_South</td></tr><tr><td>...</td></tr></table>Otherwise, the tags may end up separated from each other.<table><tr><th>Tag Name</th></tr><tr><td>North_Tank</td></tr><tr><td>...</td></tr><tr><td>...</td></tr><tr><td>...</td></tr><tr><td>South_Tank</td></tr></table>← other tags that start with the letters o, p, q, etc.</td></tr>
</table>

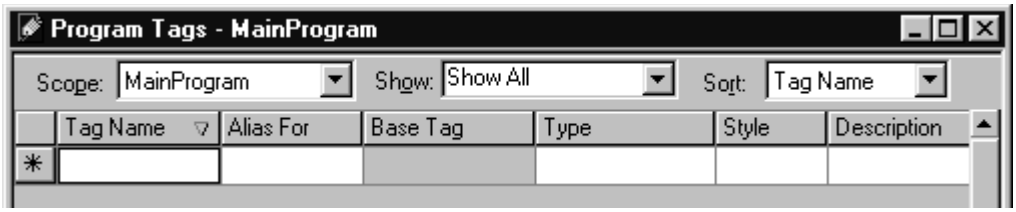
Create a Tag

IMPORTANT RSLogix 5000 software automatically creates tags when you:

- add an element to a sequential function chart (SFC)
- add a Function Block instruction to a Function Block diagram

The Tags window lets you create and edit tags using a spreadsheet-style view of the tags.

1. From the Logic menu, select Edit Tags.
2. Select a **scope** for the tag.



42350

If you will use the tag:	Then select:
in more than one program within the project	*name_of_controller(controller)*
as a producer or consumer	
in a message	
in only one program within the project	program that will use the tag

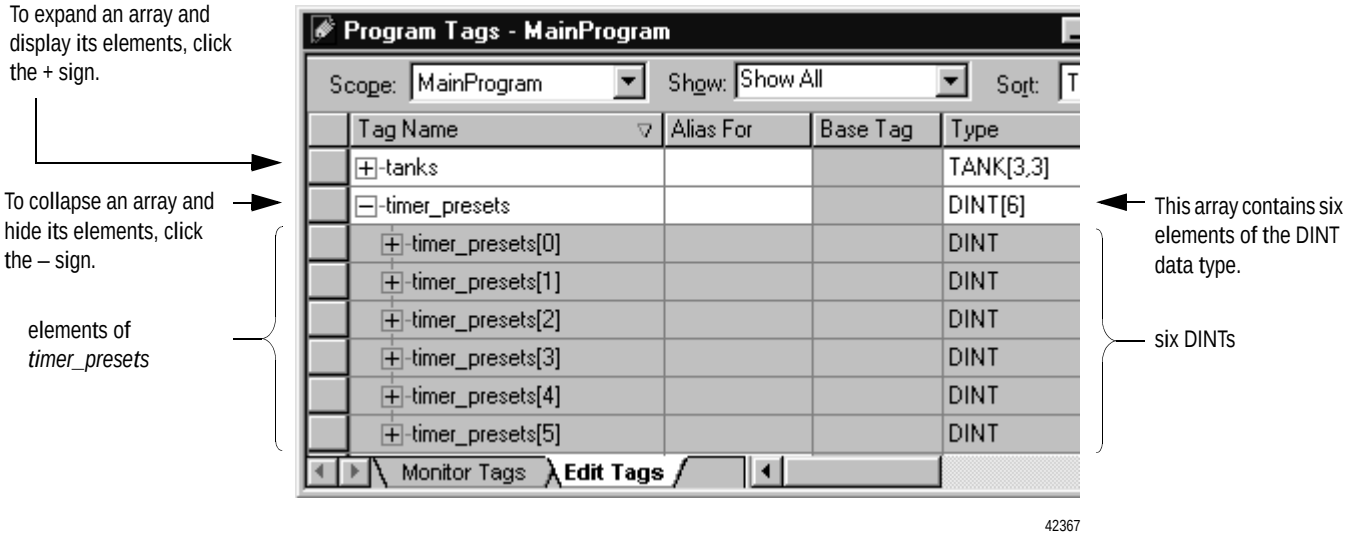
3. Type a name, data type, and description (optional) for the tag.

Create an Array

Logix5000 controllers also let you use arrays to organize data.

Term	Definition
array	A tag that contains a block of multiple pieces of data. • An array is similar to a file. • Within an array, each individual piece of data is called an element. • Each element uses the same data type. • An array tag occupies a contiguous block of memory in the controller, each element in sequence. • You can use array and sequencer instructions to manipulate or index through the elements of an array • You organize the data into a block of 1, 2, or 3 dimensions.

A subscript (s) identifies each individual **element** within the array. A subscript starts at 0 and extends to the number of elements minus 1 (zero based).



The following example compares a structure to an array:

This is a tag that uses the Timer structure (data type).

Tag Name	Data Type
⊟ Timer_1	TIMER
⊞ Timer_1.PRE	DINT
⊞ Timer_1.ACC	DINT
Timer_1.EN	BOOL
Timer_1.TT	BOOL
Timer_1.DN	BOOL

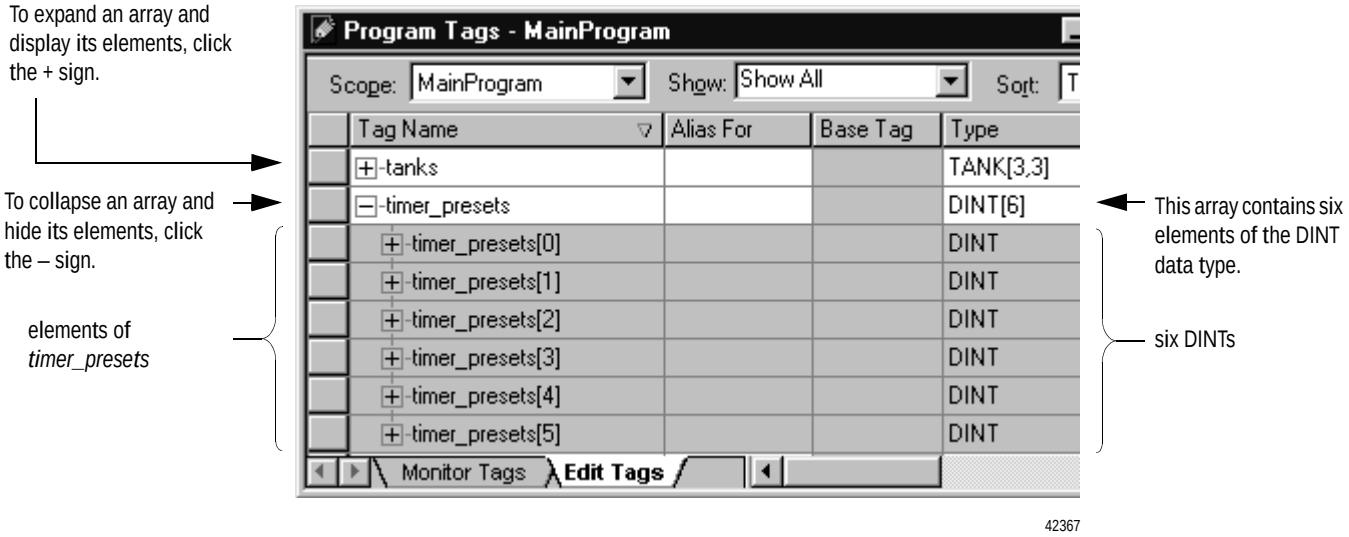
This is a tag that uses an array of the Timer data type.

Tag Name	Data Type
⊟ Timers	TIMER[3]
⊞ Timer[0]	TIMER
⊞ Timer[1]	TIMER
⊞ Timer[2]	TIMER

EXAMPLE

Single dimension array

In this example, a single timer instruction times the duration of several steps. Each step requires a different preset value. Because all the values are the same data type (DINTs) an array is used.



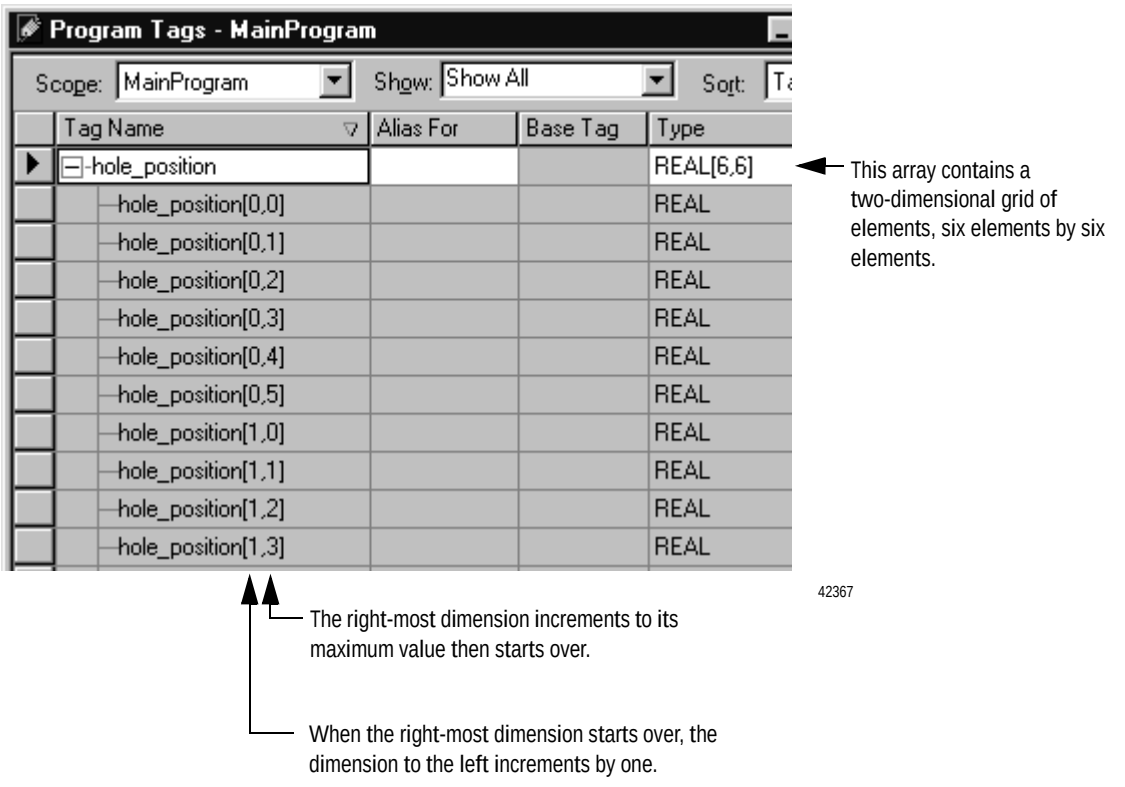
EXAMPLE

Two dimension array

A drill machine can drill one to five holes in a book. The machine requires a value for the position of each hole from the leading edge of the book. To organize the values into configurations, a two dimension array is used. The first subscript indicates the hole to which the value corresponds and the second subscript indications how many holes will be drilled (one to five).

	subscript of second dimension						Description
subscript of first dimension	0	1	2	3	4	5	
0							
1		1.5	2.5	1.25	1.25	1.25	Position of first hole from leading edge of book
2			8.0	5.5	3.5	3.5	Position of second hole from leading edge of book
3				9.75	7.5	5.5	Position of third hole from leading edge of book
4					9.75	7.5	Position of fourth hole from leading edge of book
5						9.75	Position of fifth hole from leading edge of book

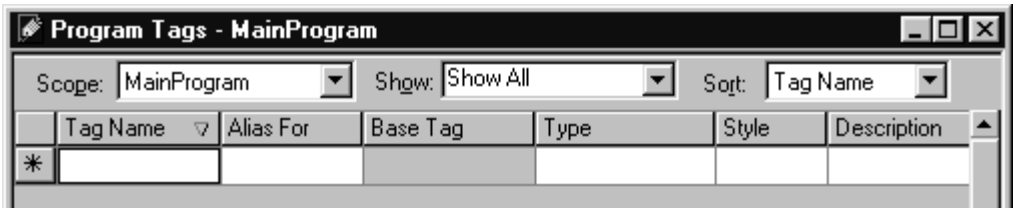
In the Tags window, the elements are in the order depicted below.



Create an Array

To create an array, you create a tag and assign dimensions to the data type:

1. From the Logic menu, select Edit Tags.



42350

2. Type a **name** for the tag and select a **scope** for the tag:

If you will use the tag	Then select
in more than one program within the project	*name_of_controller*(*controller*)
as a producer or consumer	
in a message	
in only one program within the project	program that will use the tag

3. Assign the array dimensions:

If the tag is	Then type	Where
one **dimension** array	*data_type*[*x*]	*data_type* is the type of data that the tag stores. *x* is the number of **elements** in the first dimension. *y* is the number of elements in the second dimension. *z* is the number of elements in the third dimension.
two dimension array	*data_type*[*x*,*y*]	
three dimension array	*data_type*[*x*,*y*,*z*]	

Create a User-Defined Data Type

User-defined data types (structures) let you organize your data to match your machine or process.

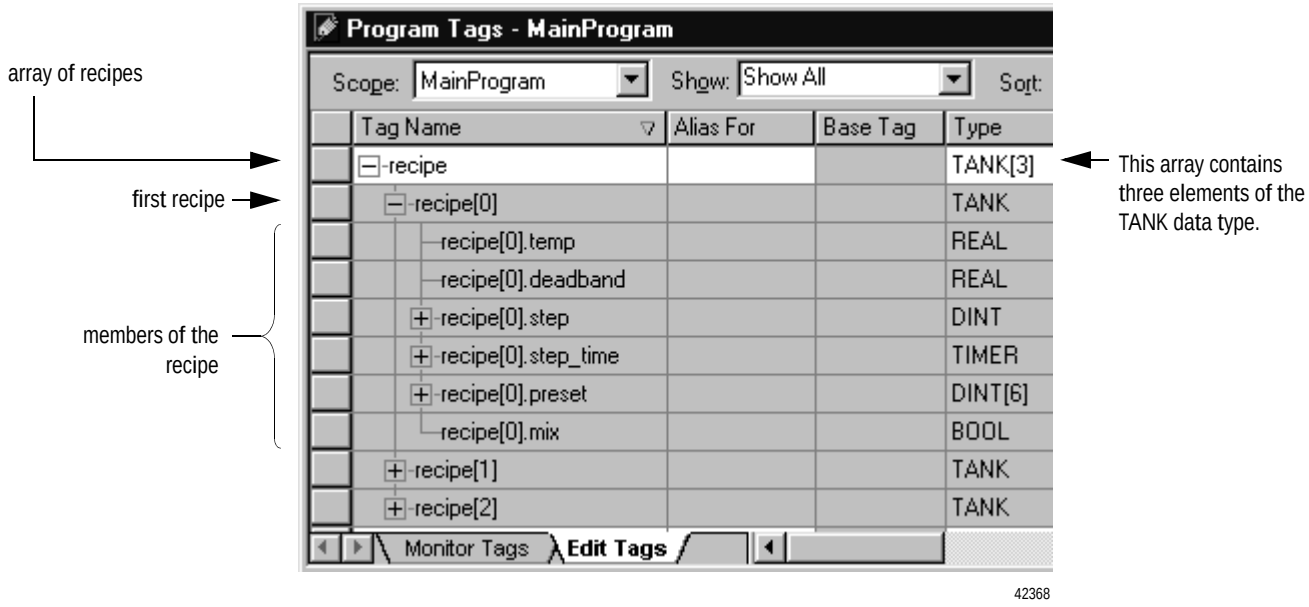
EXAMPLE

User-defined data type that stores a recipe

In a system of several tanks, each tank can run a variety of recipes. Because the recipe requires a mix of data types (REAL, DINT, BOOL, etc.) a user-defined data type is used.

Name (of data type): TANK	
Member Name	**Data Type**
temp	REAL
deadband	REAL
step	DINT
step_time	TIMER
preset	DINT[6]
mix	BOOL

An array that is based on this data type would look like this:



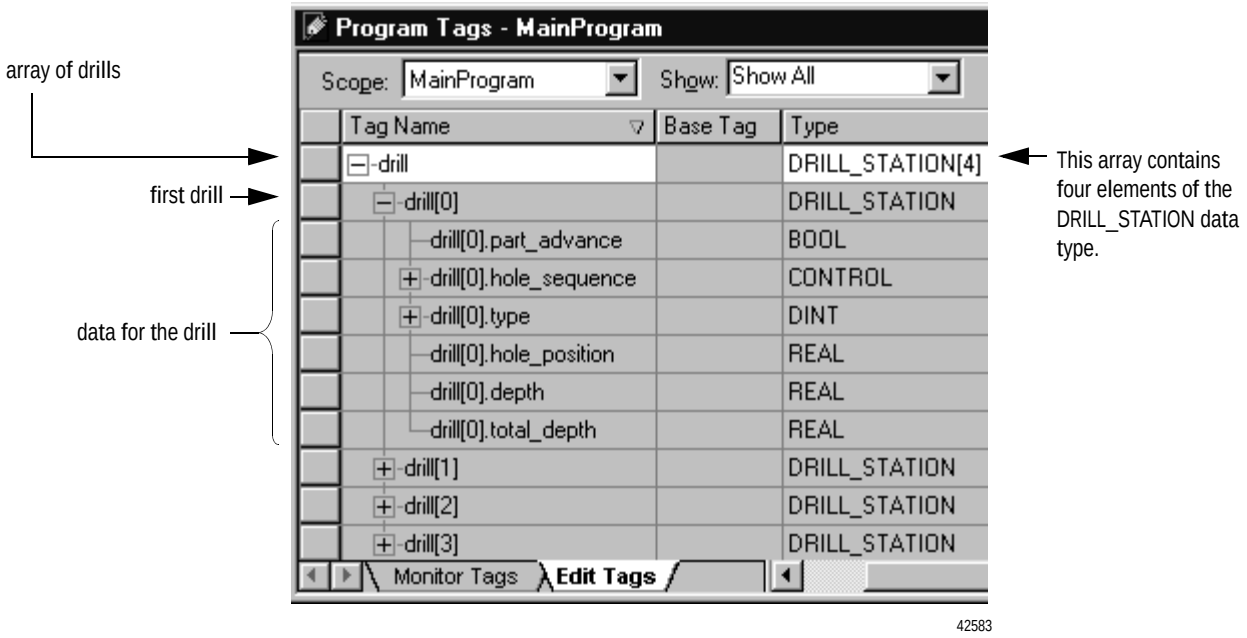
EXAMPLE

User-defined data type that stores the data that is required to run a machine

Because several drill stations require the following mix of data, use a user-defined data type

Name (of data type): DRILL_STATION	
Member Name	**Data Type**
part_advance	BOOL
hole_sequence	CONTROL
type	DINT
hole_position	REAL
depth	REAL
total_depth	REAL

An array that is based on this data type looks like:



Guidelines for User-Defined Data Types

When you create a user-defined data type, keep the following in mind:

- If you include members that represent I/O devices, you must use logic to copy the data between the members in the structure and the corresponding I/O tags. Refer to "Address I/O Data" on page 27.
- If you include an array as a member, limit the array to a single dimension. Multi-dimension arrays are *not* permitted in a user-defined data type.
- When you use the BOOL, SINT, or INT data types, place members that use the same data type in sequence:

more efficient

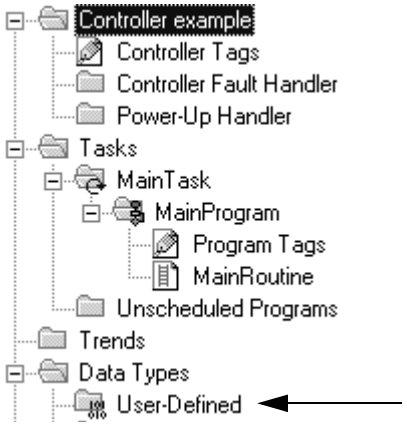
BOOL
BOOL
BOOL
DINT
DINT

less efficient

BOOL
DINT
BOOL
DINT
BOOL

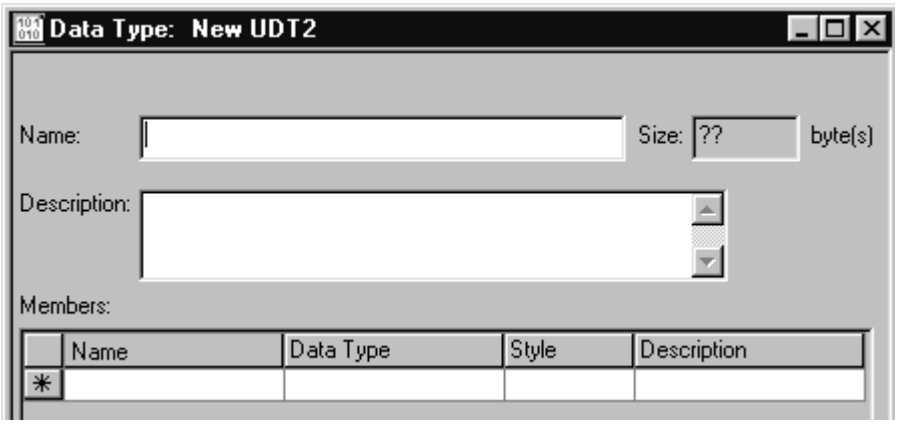
Create a User-Defined Data Type

1. From the User-Defined folder under Data Types, right-click User-Defined and select New Data Type.



2. Type a **name** for the array, **description** for the array.

3. For each **member** of the array, type a name, data type, style, and description (optional).



42196

Limit any arrays to a single dimension.

To display the value (s) of the member in a different **style** (radix), select the style.

4. Click Apply.

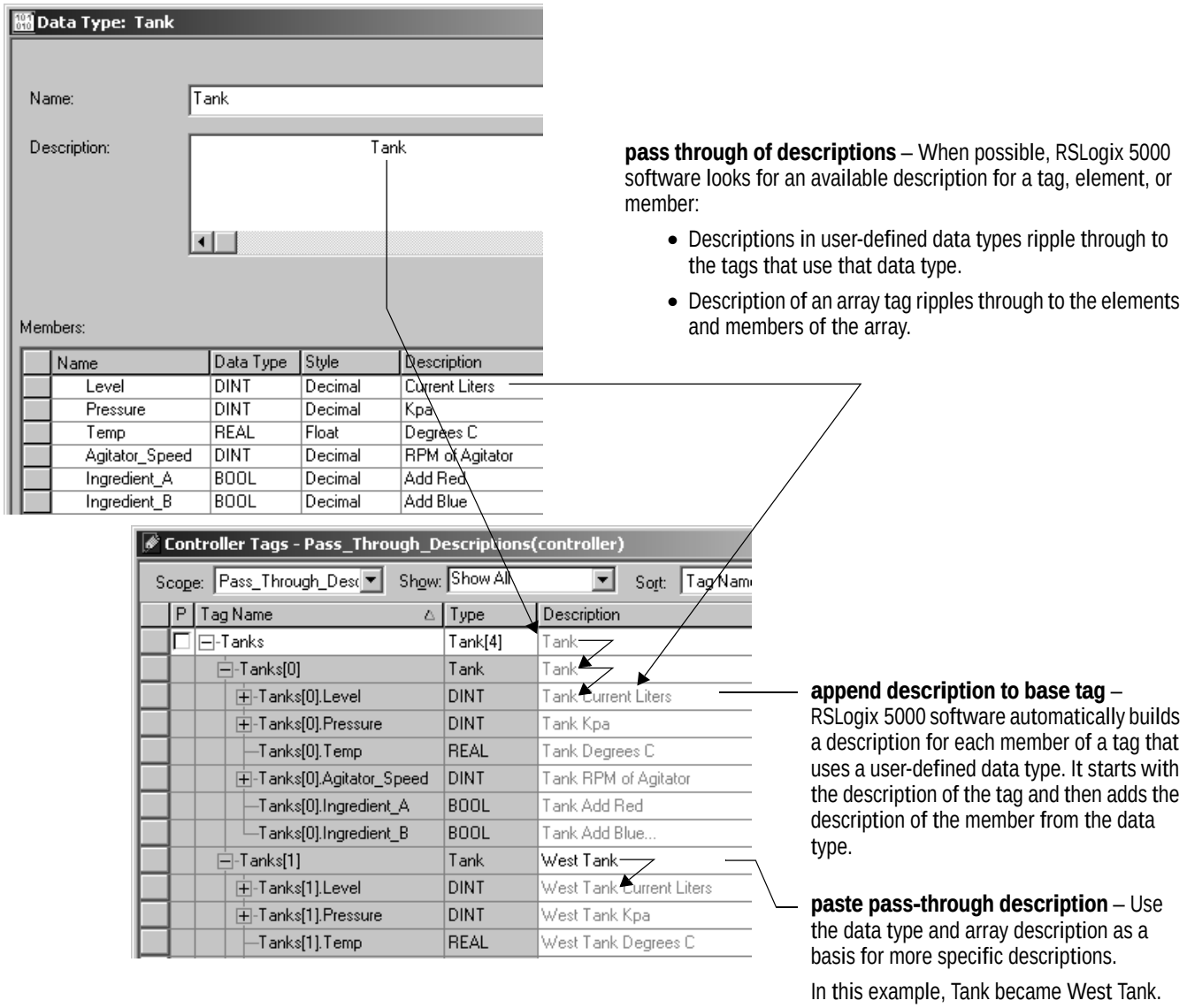
5. Add as many members as needed.

Describe a User-Defined Data Type

RSLogix 5000 software
13.0 or later

RSLogix 5000 software lets you automatically build descriptions out of the descriptions in your user-defined data types. This greatly reduces the amount of time you have to spend documenting your project.

As you organize your user-defined data types, keep in mind the following features of RSLogix 5000 software:



pass through of descriptions – When possible, RSLogix 5000 software looks for an available description for a tag, element, or member:

- Descriptions in user-defined data types ripple through to the tags that use that data type.
- Description of an array tag ripples through to the elements and members of the array.

append description to base tag – RSLogix 5000 software automatically builds a description for each member of a tag that uses a user-defined data type. It starts with the description of the tag and then adds the description of the member from the data type.

paste pass-through description – Use the data type and array description as a basis for more specific descriptions.

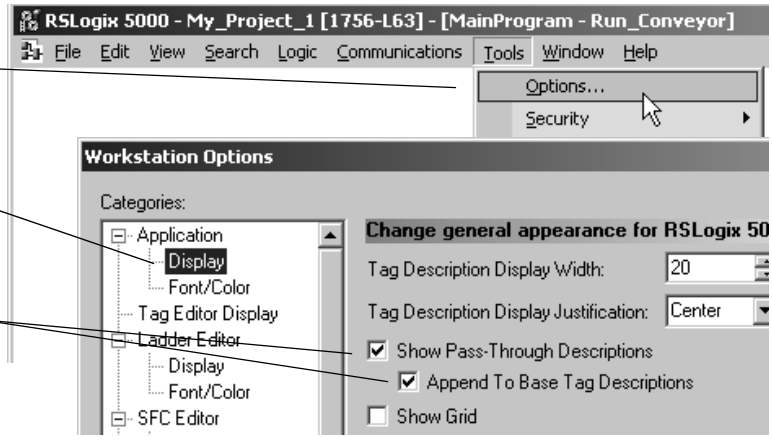
In this example, Tank became West Tank.

RSLogix 5000 software uses different colors for descriptions:

A description in this color	Is a
gray	pass-through description
black	manually entered description

Turn Pass-Through and Append Descriptions On or Off

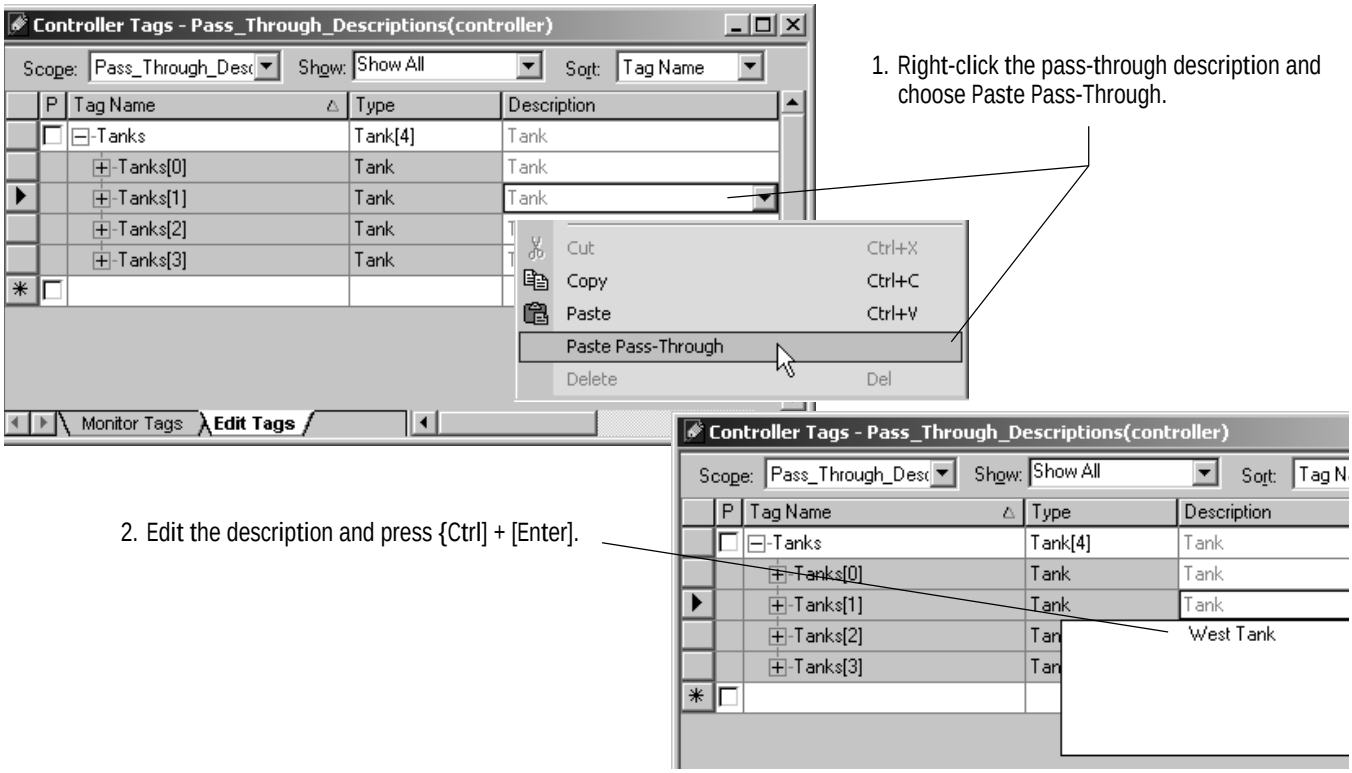
1. In RSLogix 5000 software, choose Tools > Options.
2. Select the Application > Display.
3. Turn on (check) or turn off (uncheck) the desired options.



Paste a Pass-Through Description

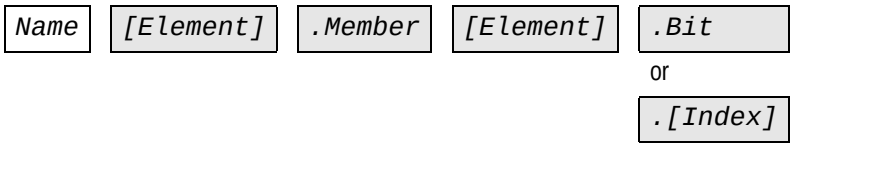
To use a pass-through description as the starting point for a more specific description:

1. Right-click the pass-through description and choose Paste Pass-Through.
2. Edit the description and press {Ctrl} + [Enter].



Address Tag Data

An tag name follows this format:



Where	Is
Name	Name that identifies this specific tag.
Element	Subscript or subscripts that point to a specific element within an array. • Use the element identifier only if the tag or member is an array. • Use one subscript for each dimension of the array. For example: [5], [2,8], [3,2,7]. To indirectly (dynamically) reference an element, use a tag or numeric expression that provides the element number. • A numeric expression uses a combination of tags, constants, operators, and functions to calculate a value. For example, `Tag_1-Tag_2`, `Tag_3+4`, `ABS(Tag_4)`. • Keep the value of the tag or numeric expression within the dimensions of the array. For example, if a dimension of an array contains 10 elements, then the value of the tag or numeric expression must be 0 to 9 (10 elements).
Member	Specific member of a structure. • Use the member identifier only if the tag is a structure. • If the structure contains another structure as one of its members, use additional levels of the `.Member` format to identify the required member.
Bit	Specific bit of an integer data type (SINT, INT, or DINT).
Index	To indirectly (dynamically) reference a bit of an integer, use a tag or numeric expression that provides the bit number. • A numeric expression uses a combination of tags, constants, operators, and functions to calculate a value. For example, `Tag_1-Tag_2`, `Tag_3+4`, `ABS(Tag_4)`. • Keep the value of the tag or numeric expression within the range of bits of the integer tag. For example, if the integer tag is a Dint (32-bits), then the value of the index must be 0 to 31 (32-bits).

Assigning Alias Tags

An alias tag lets you create one tag that represents another tag.

- Both tags share the same value (s).
- When the value (s) of one of the tags changes, the other tag reflects the change as well.

Use aliases in the following situations:

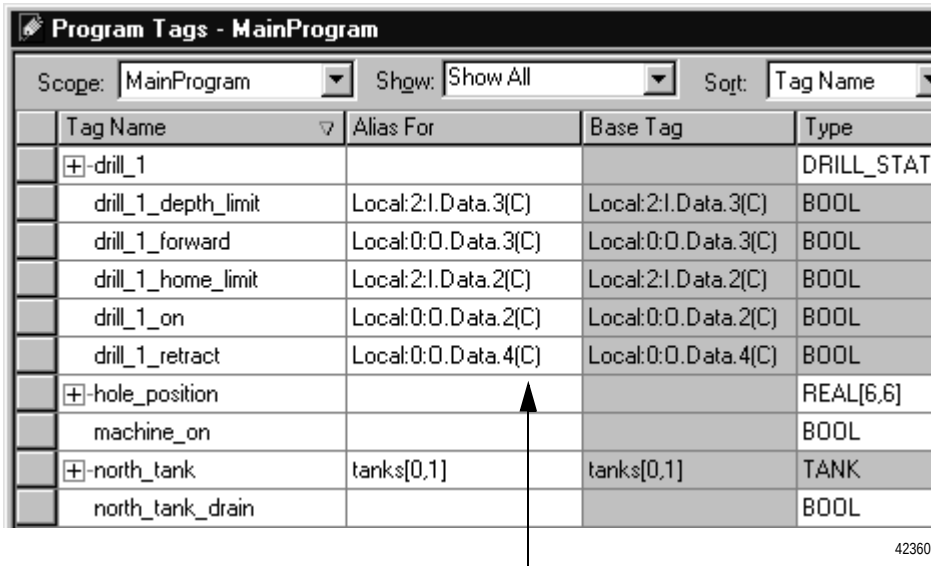
- program logic in advance of wiring diagrams
- assign a descriptive name to an I/O device
- provide a more simple name for a complex tag
- use a descriptive name for an element of an array

The tags window displays alias information.

drill_1_depth_limit is an alias for *Local:2:I.Data.3* (a digital input point). When the input turns on, the alias tag also turns on.

drill_1_on is an alias for *Local:0:O.Data.2* (a digital output point). When the alias tag turns on, the output tag also turns on.

north_tank is an alias for *tanks[0,1].*

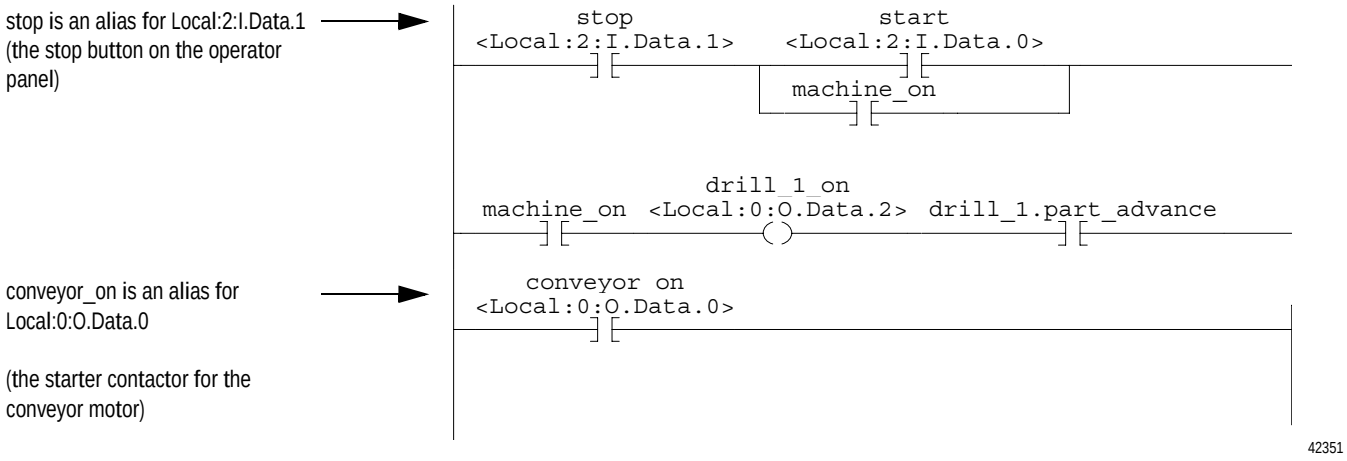


The *(C)* indicates that the tag is at the controller scope.

A common use of alias tags is to program logic before wiring diagrams are available:

1. For each I/O device, create a tag with a name that describes the device, such as *conveyor* for the conveyor motor.
2. Program your logic using the descriptive tag names. (You can even test your logic without connecting to the I/O.)
3. Later, when wiring diagrams are available, add the I/O modules to the I/O configuration of the controller.
4. Finally, convert the descriptive tags to aliases for their respective I/O points or channels.

The following logic was initially programmed using descriptive tag names, such as stop and conveyor_on. Later, the tags were converted to aliases for the corresponding I/O devices.



Display Alias Information

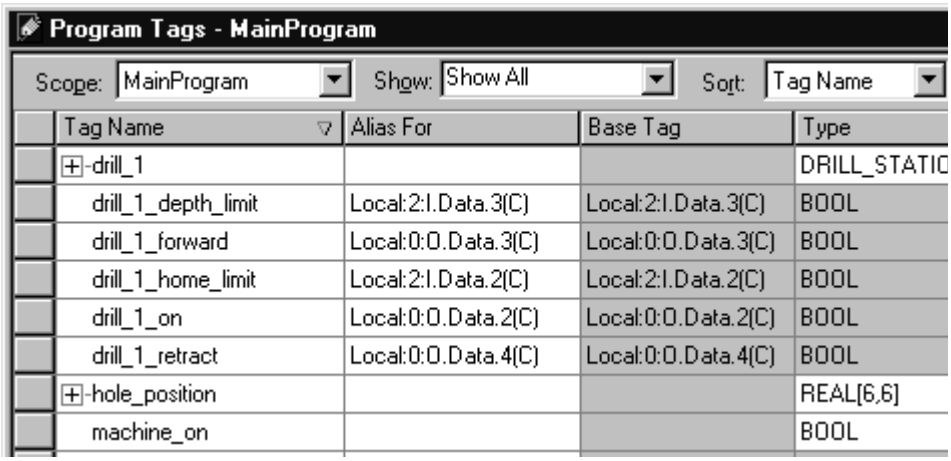
To show (in your logic) the tag to which an alias points, do the following:

1. From the Tools menu, select Options.
2. Select the Ladder Display tab.
3. Check the Show Tag Alias Information check box.
4. Click OK.

Assign an Alias

To assign a tag as an **alias tag** for another tag:

1. From the Logic menu, select Edit Tags.



2. Select the **scope** of the tag.
3. To the right of the tag name, click the Alias For cell.

   The cell displays a t
4. Click the t
5. Select the tag that the alias will represent:

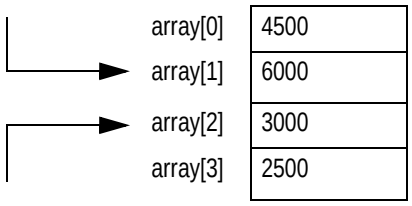
To	Do this
select a tag	Double-click the tag name.
select a bit number	A. Click the tag name. B. To the right of the tag name, click t C. Click the required bit.

6. Press [Enter] or click another cell.

Assigning an Indirect Address

If you want an instruction to access different elements in an array, use a tag in the subscript of the array (an indirect address). By changing the value of the tag, you change the element of the array that your logic references.

When index equals 1, array[index] points here.



When index equals 2, array[index] points here.

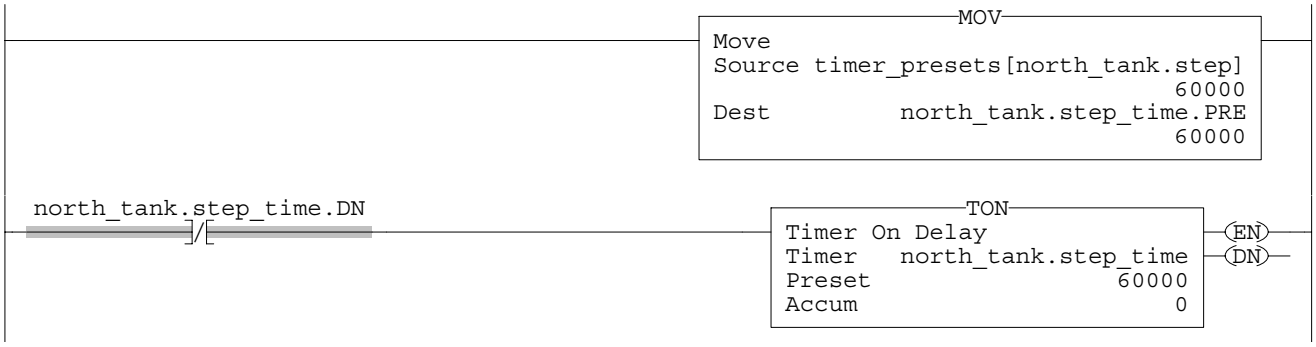
The following table outlines some common uses for an indirect address:

To	Use a tag in the subscript and
select a recipe from an array of recipes	Enter the number of the recipe in the tag.
load a specific machine setup from an array of possible setups	Enter the desired setup in the tag.
load parameters or states from an array, one element at a time	A. Perform the required action on the first element. B. Use an ADD instruction to increment the tag value and point to the next element in the array.
log error codes	
perform several actions on an array element and then index to the next element	

The following example loads a series of preset values into a timer, one value (array element) at a time.

EXAMPLE Step through an array

The timer_presets array stores a series of preset values for the timer in the next rung. The north_tank.step tag points to which element of the array to use. For example, when north_tank.step equals 0, the instruction loads timer_presets[0] into the timer (60,000 ms).



When north_tank.step_time is done, the rung increments north_tank.step to the next number and that element of the timer_presets array loads into the timer.



When north_tank.step exceeds the size of the array, the rung resets the tag to start at the first element in the array. (The array contains elements 0 to 3.)



42358

Expressions

You can also use an expression to specify the subscript of an array.

- An expression uses operators, such as + or -, to calculate a value.
- The controller computes the result of the expression and uses it as the array subscript.

You can use these operators to specify the subscript of an array:

Operator	Description
+	add
-	subtract/negate
*	multiply
/	divide
ABS	Absolute value
AND	AND
FRD	BCD to integer

Operator	Description
MOD	Modulo
NOT	complement
OR	OR
SQR	square root
TOD	integer to BCD
TRN	Truncate
XOR	exclusive OR

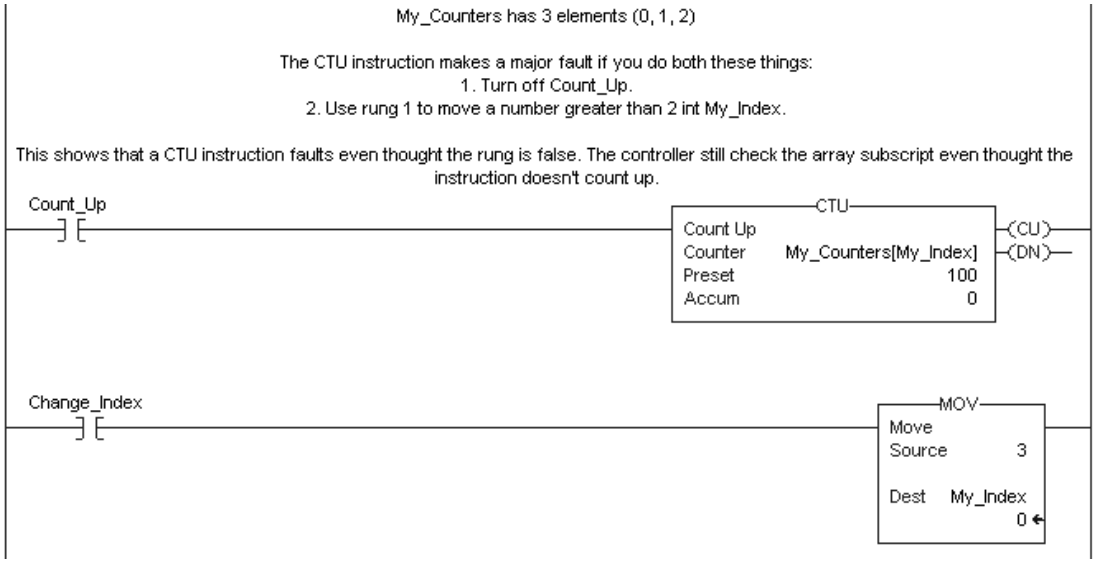
Format your expressions as follows:

If the operator requires	Use this format	Examples
one value (tag or expression)	*operator(value)*	ABS(tag_a)
two values (tags, constants, or expressions)	*value_a operator value_b*	• tag_b + 5 • tag_c AND tag_d • (tag_e ** 2) MOD (tag_f / tag_g)

Array Subscript Out of Range

Every instruction generates a major fault if the array subscript is out of range. Transitional instructions also generate a major fault even if the rung is false. The controller checks the array subscript in these instructions even if the rung is false.

EXAMPLE



For more information on handling major faults, see chapter 16 Handle a Major Fault.

Notes:

Managing Tasks

Introduction

The default RSLogix 5000 project provides a single task for all your logic. While this is sufficient for many applications, some situations may require more than one task.

This chapter helps you manage tasks in your project:

Topic	Page
Select Controller Tasks	60
Prioritize Periodic and Event Tasks	62
Leave Enough Time for Unscheduled Communication	65
Avoid Overlaps	66
Configuring Output Processing for a Task	70
Inhibit a Task	74
Choose the Trigger for an Event Task	77
Using the Module Input Data State Change Trigger	79
Using the Motion Group Trigger	88
Using the Axis Registration Trigger	90
Using the Axis Watch Trigger	94
Using the Consumed Tag Trigger	98
Using the EVENT Instruction Trigger	106
Define a Timeout Value for an Event Task	111
Create a Task	109
Adjust the System Overhead Time Slice	115
Adjust the Watchdog Time	118

Select Controller Tasks

A Logix5000 controller supports multiple tasks to schedule and prioritize the execution of your programs based on specific criteria. This balances the processing time of the controller.

- The controller executes only one task at one time.
- A different task can interrupt a task that is executing and take control.
- In any given task, only one program executes at one time.

A Logix5000 controller supports three types of tasks:

If you want to execute a section of your logic:	Then use this type of task:	Description:
all of the time	Continuous Task	The continuous task runs in the background. Any CPU time not allocated to other operations (such as motion, communications, and periodic or event tasks) is used to execute the programs within the continuous task. • The continuous task runs all the time. When the continuous task completes a full scan, it restarts immediately. • A project does not require a continuous task. If used, there can be only one continuous task.
• at a constant period (e.g., every 100 ms) • multiple times within the scan of your other logic	Periodic Task	A periodic task performs a function at a specific period. Whenever the time for the periodic task expires, the periodic task: • interrupts any lower priority tasks • executes one time • returns control to where the previous task left off You can configure the time period from 0.1 ms to 2000 s. • The default is 10 ms. • The performance of a periodic task depends on the type of Logix5000 controller and on the logic in the task.
immediately when an event occurs	Event Task	An event task performs a function only when a specific event (trigger) occurs. Whenever the trigger for the event task occurs, the event task: • interrupts any lower priority tasks • executes one time • returns control to where the previous task left off The trigger can be: • change of a digital input • new sample of analog data • certain motion operations • consumed tag • EVENT instruction **Important:** Some Logix5000 controllers do not support all triggers. See Table 3.1 on page 3-78.

Here are some example situations for the tasks:

For this example situation	Use this type of task
Fill a tank to its maximum level and then open a drain valve	continuous task
Collect and process system parameters and send them to a display	continuous task
Complete step 3 in a control sequence—reposition the bin diverter	continuous task
Your system must check the position of a field arm each 0.1 s and calculate the average rate of change in its position. This is used to determine braking pressure.	periodic task
Read the thickness of a paper roll every 20 ms.	periodic task
A packaging line glues boxes closed. When a box arrives at the gluing position, the controller must immediately execute the gluing routine.	event task
In a high-speed assembly operation, an optical sensor detects a certain type of reject. When the sensor detects a reject, the machine must immediately divert the reject.	event task
In an engine test stand, you want to capture and archive each analog data immediately after each sample of data	event task
Immediately after receiving new production data, load the data into the station	event task
In a line that packages candy bars, you have to make sure that the perforation occurs in the correct location on each bar. Each time the registration sensor detects the registration mark, check the accuracy of an axis and perform any required adjustment.	event task
A gluing station must adjust the amount of glue it applies to compensate for changes in the speed of the axis. After the motion planner executes, check the command speed of the axis and vary the amount of glue, if needed.	event task
In a production line, if any of the programs detect an unsafe condition the entire line must shut down. The shutdown procedure is the same regardless of the unsafe condition.	event task

The number of tasks supported depends on the controller:

This controller	Supports this number of tasks	Notes
ControlLogix SoftLogix5800	32	Only one task can be continuous.
CompactLogix DriveLogix FlexLogix	8	

Use Caution in the Number of Tasks That You Use

Typically, each task takes controller time away from the other tasks. If you have too many tasks, then:

- The continuous task may take too long to complete.
- Other tasks may experience overlaps. If a task is interrupted too frequently or too long, it may not complete its execution before it is triggered again.

For more information, see "Avoid Overlaps" on page 66.

Prioritize Periodic and Event Tasks

Although a project can contain multiple tasks, the controller executes only one task at a time. If a periodic or event task is triggered while another task is currently executing, the priority of each task tells the controller what to do.

The number of priority levels depends on the controller:

This Logix5000 controller	Has this many priority levels
CompactLogix	15
ControlLogix	15
DriveLogix	15
FlexLogix	15
SoftLogix5800	3

To assign a priority to a task, use these guidelines:

If you want	Then	Notes
this task to interrupt another task	Assign a priority number that is less than (higher priority) the priority number of the other task.	• A higher priority task interrupts all lower priority tasks. • A higher priority task can interrupt a lower priority task multiple times.
another task to interrupt this task	Assign a priority number that is greater than (lower priority) the priority number of the other task.	
this task to share controller time with another task	Assign the same priority number to both tasks.	The controller switches back and forth between each task and executes each one for 1 ms.

Additional Considerations

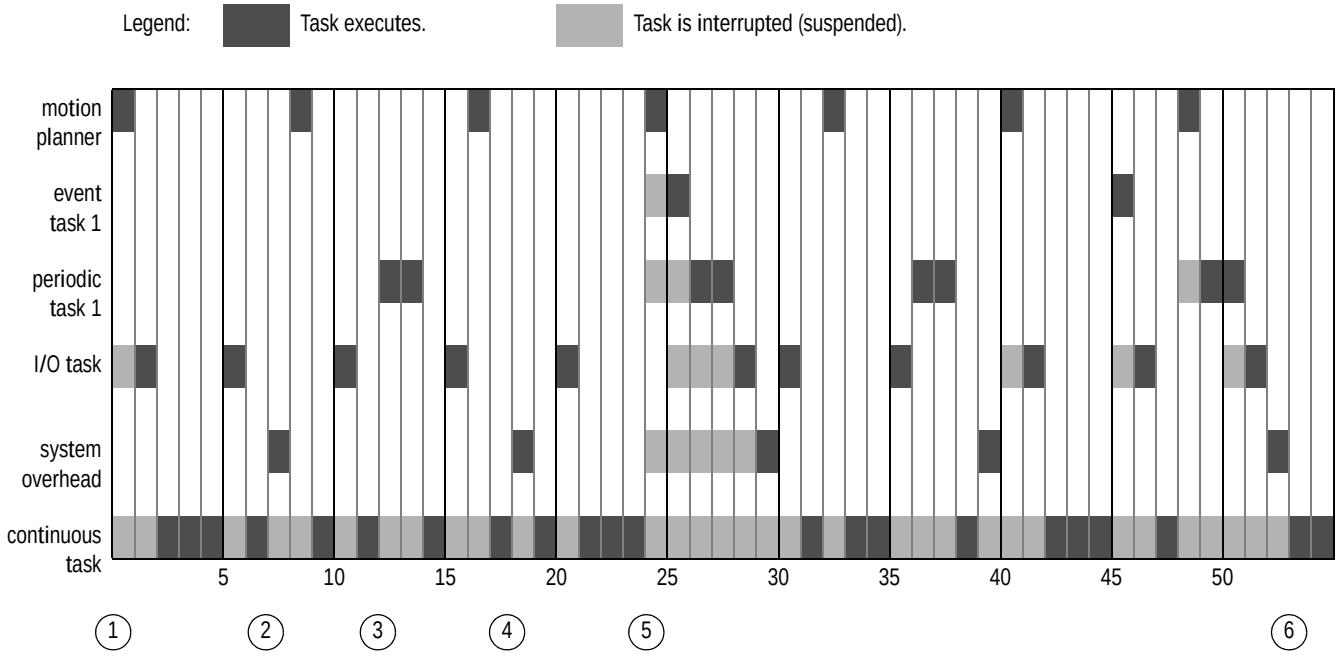
As you estimate the execution interrupts for a task, consider:

<table>
<tr><th>Consideration</th><th>Description</th></tr>
<tr><td>motion planner</td><td>The motion planner interrupts all other tasks, regardless of their priority.
The number of axes and coarse update period for the motion group effect how long and how often the motion planner executes.
If the motion planner is executing when a task is triggered, the task waits until the motion planner is done.
If the coarse update period occurs while a task is executing, the task pauses to let the motion planner execute.</td></tr>
<tr><td>I/O task</td><td>CompactLogix, FlexLogix, and DriveLogix controllers use a dedicated periodic task to process I/O data. This I/O task:
Does not show up in the Tasks folder of the controller.
Does not count toward the task limits for the controller.
Operates at priority 7.
Executes at the fastest RPI you have scheduled for the system.
Executes for as long as it takes to scan the configured I/O modules.
As you assign priorities to your tasks, consider the I/O task:<table><tr><th>If you want a task to</th><th>Then assign one of these priorities</th></tr><tr><td>interrupt or delay I/O processing</td><td>1 to 6</td></tr><tr><td>share controller time with I/O processing</td><td>7</td></tr><tr><td>let I/O processing interrupt or delay the task</td><td>8 to 15</td></tr></table></td></tr>
<tr><td>system overhead</td><td>System overhead is the time that the controller spends on unscheduled communication.
Unscheduled communication is any communication that you do not configure through the I/O configuration folder of the project, such as Message (MSG) instructions and communication with HMIs or workstations.
System overhead interrupts only the continuous task.
The system overhead time slice specifies the percentage of time (excluding the time for periodic or event tasks) that the controller devotes to unscheduled communication.
The controller performs unscheduled communication for up to 1 ms at a time and then resumes the continuous task.</td></tr>
<tr><td>continuous task</td><td>You do not assign a priority to the continuous task. It always runs at the lowest priority. All other tasks interrupt the continuous task.</td></tr>
</table>

EXAMPLE

This example depicts the execution of a project with three user tasks.

Task	Priority	Period	Execution time	Duration
motion planner	n/a	8 ms (course update rate)	1 ms	1 ms
event task 1	1	n/a	1 ms	1 to 2 ms
periodic task 1	2	12 ms	2 ms	2 to 4 ms
I/O task—n/a to ControlLogix and SoftLogix controllers. See page 63.	7	5 ms (fastest RPI)	1 ms	1 to 5 ms
system overhead	n/a	time slice = 20%	1 ms	1 to 6 ms
continuous task	n/a	n/a	20 ms	48 ms



Description	
1	Initially, the controller executes the motion planner and the I/O task (if one exists).
2	After executing the continuous task for 4 ms, the controller triggers the system overhead.
3	The period for periodic task 1 expires (12 ms), so the task interrupts the continuous task.
4	After executing the continuous task again for 4 ms, the controller triggers the system overhead.
5	The triggers occurs for event task 1. Event task 1 waits until the motion planner is done. Lower priority tasks experience longer delays.
6	The continuous task automatically restarts.

RSLogix 5000 software includes a task monitor tool on the distribution CD. You can use this tool to analyze how tasks are executing.

Leave Enough Time for Unscheduled Communication

Unscheduled communication occurs only when a periodic or event task is not running. If you use multiple tasks, make sure that their scan times and execution intervals leave enough time for unscheduled communication.

1. The execution time of a highest priority task is significantly less than its update rate.

2. The total execution time of all your tasks is significantly less than the update rate of the lowest priority tasks.

For example, in this configuration of tasks:

Task	Priority	Execution time	Rate
1	higher	20 ms	80 ms
2	lower	30 ms	100 ms
	total execution time:	50 ms	

- The execution time of the highest priority task (Task 1) is significantly less than its update rate (20 ms is less than 80 ms).
- The total execution time of all tasks is significantly less than the update rate of the lowest priority task (50 ms is less than 100 ms).

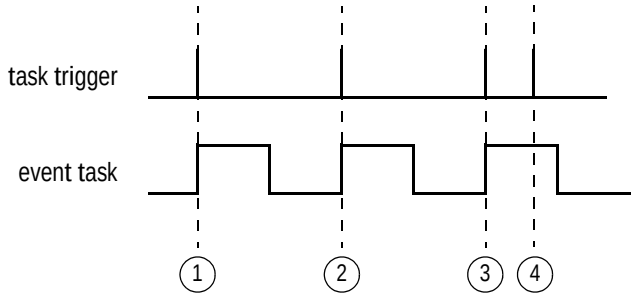
This generally leaves enough time for unscheduled communication.

- Adjust the update rates of the tasks as needed to get the best trade-off between executing your logic and servicing unscheduled communication.
- If your project has a continuous task, unscheduled communication occurs as a percentage of controller time (excluding the time for periodic or event tasks). See “system overhead” on page 63.

Avoid Overlaps

An **overlap** is a condition where a task (periodic or event) is triggered while the task is still executing from the previous trigger.

IMPORTANT If an overlap occurs, the controller disregards the trigger that caused the overlap. In other words, you might miss an important execution of the task.



	Description
1	Task trigger occurs. Task executes.
2	Task trigger occurs. Task executes.
3	Task trigger occurs. Task executes.
4	Overlap occurs. Task is triggered while it is still executing. The trigger does not restart the task. The trigger is ignored.

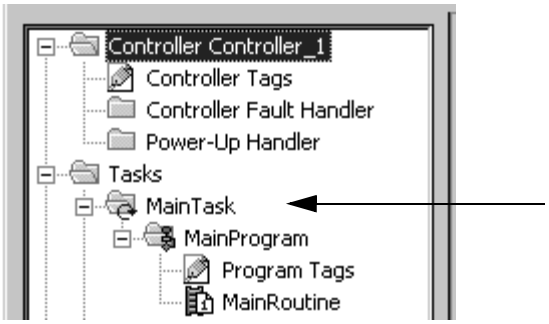
Each task requires enough time to finish before it is triggered again. Make sure that the scan time of the task is significantly less than the rate at which the trigger occurs. If an overlap occurs, reduce the frequency at which you trigger the task:

If the type of task is	Then
periodic	increase the period of the task
event	adjust the configuration of your system to trigger the task less frequently.

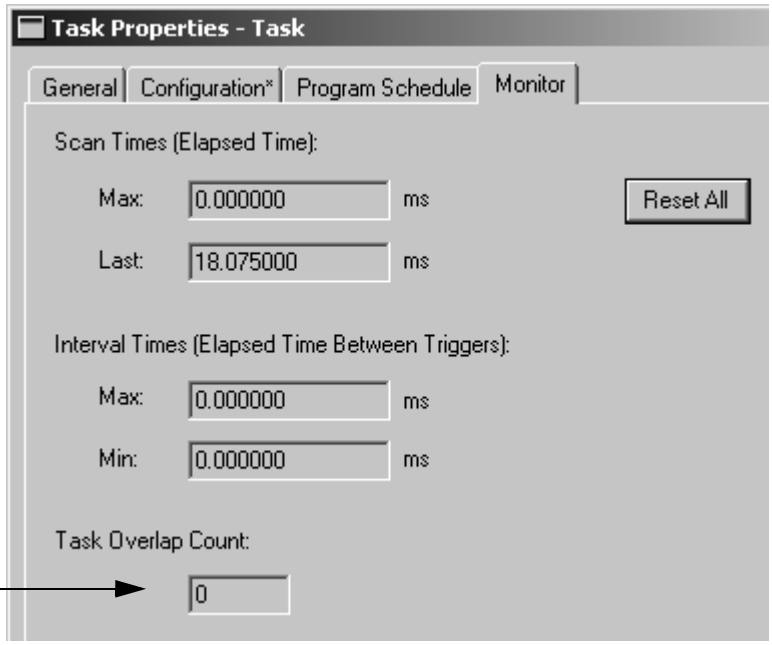
Manually Check for Overlaps

To manually see if overlaps are occurring for a task:

1. In the controller organizer, right-click the task and select Properties.



2. Select the Monitor tab.



3. Click OK.

Programmatically Check for Overlaps

When an overlap occurs, the controller:

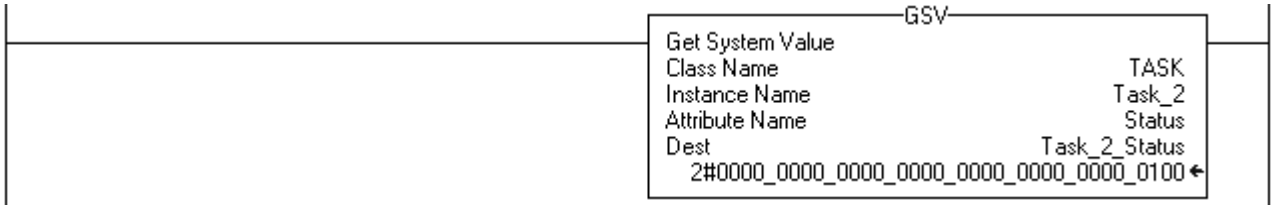
- logs a minor fault to the FAULTLOG object
- stores overlap information in the TASK object for the task

To write logic to check for an overlap, use a Get System Value (GSV) instruction to monitor either of these objects:

If you want to	Then access the object and attribute			
	Object	**Attribute**	**Data Type**	**Description**
determine if an overlap occurred for any task	FAULTLOG	MinorFaultBits	DINT	Individual bits that indicate a minor fault:
				To determine if: / **Examine this bit:**
				An instruction produced a minor fault. / 4
				An overlap occurred for a task. / 6
				The serial port produced a minor fault. / 9
				The battery is not present or needs replacement. / 10
determine if an overlap occurred for a specific task	TASK	Status	DINT	Status information about the task. Once the controller sets one of these bits, you must manually clear the bit.
				To determine if: / **Examine this bit:**
				An EVENT instruction triggered the task (event task only). / 0
				A timeout triggered the task (event task only). / 1
				An overlap occurred for this task. / 2
determine the number of times that an overlap occurred.	TASK	OverlapCount	DINT	Valid for an event or a periodic task. To clear the count, set the attribute to 0.

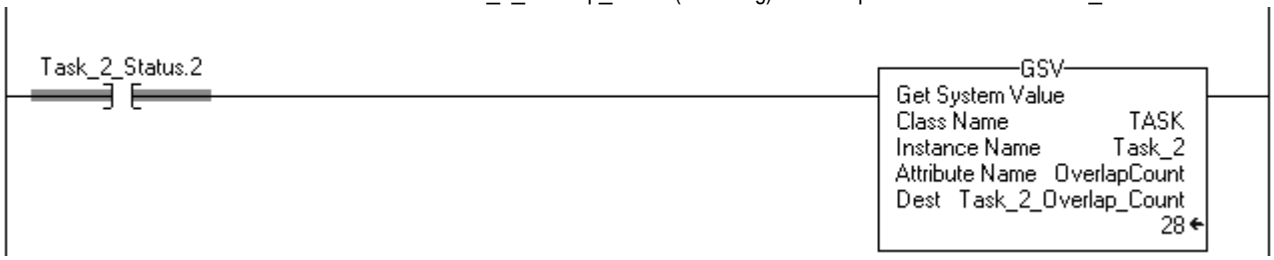
EXAMPLE Programmatically Check for Overlaps

1. The GSV instruction sets Task_2_Status = Status attribute for Task_2 (DINT value).



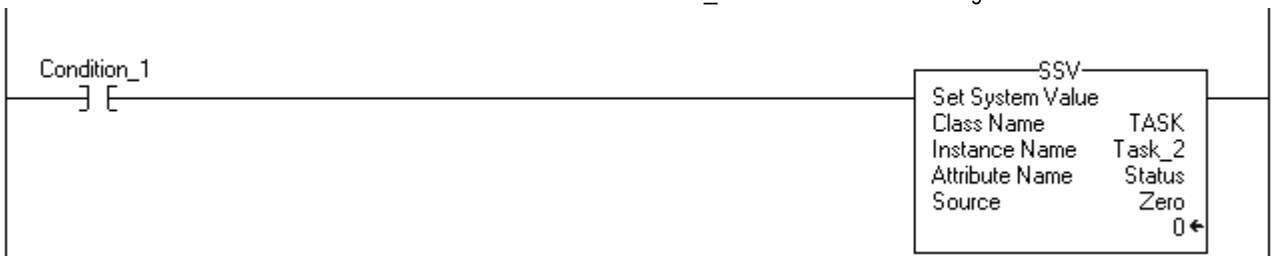
2. If Task_2_Status.2 = 1, then an overlap occurred so get the count of overlaps:

   The GSV instruction sets Task_2_Overlap_Count (DINT tag) = OverlapCount attribute of Task_2.



3. If Condition_1 = 1, then clear the bits of the Status attribute for Task_2:

   The SSV instruction sets the Status attribute of Task_2 = Zero. Zero is a DINT tag with a value of 0.

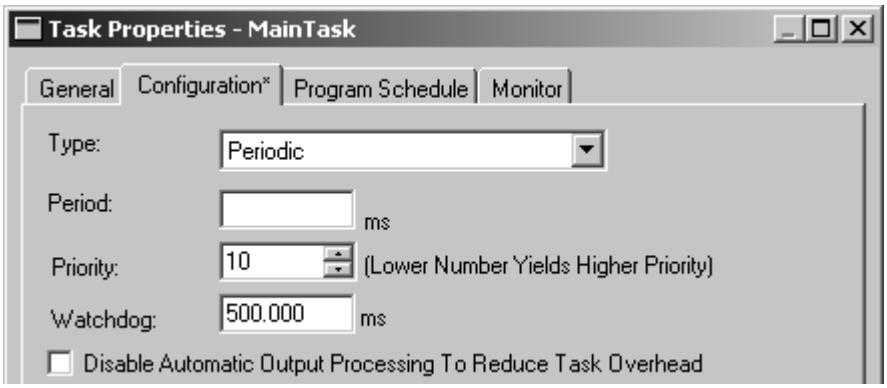


Configuring Output Processing for a Task

At the end of a task, the controller performs overhead operations (output processing) for the I/O modules in your system. While not the same as updating the modules, this output processing may effect the update of the I/O modules in your system.

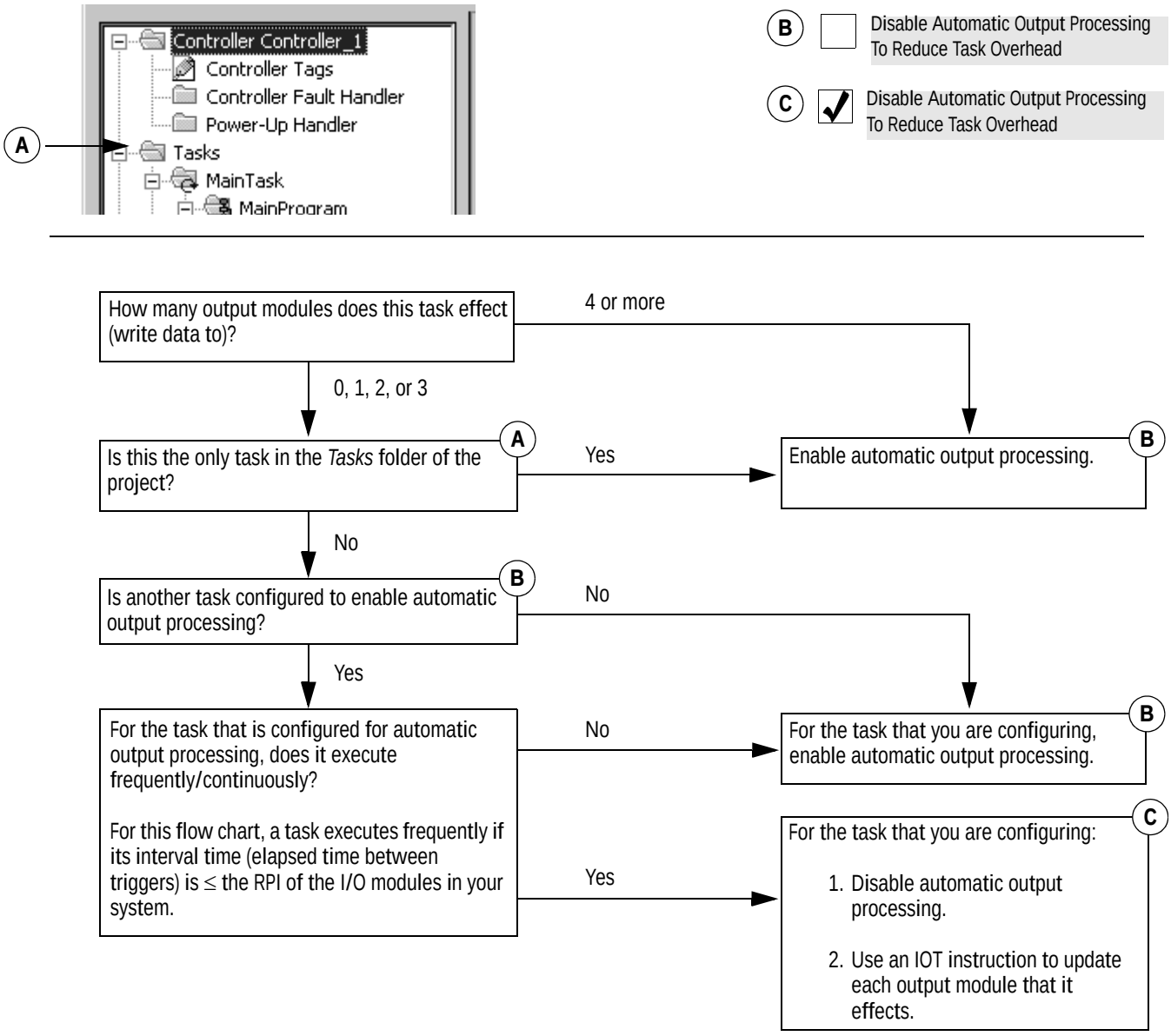
As an option, you can turn off this output processing for a specific task, which reduces the elapsed time of that task.

Enable or disable the processing of outputs at the end of the task →



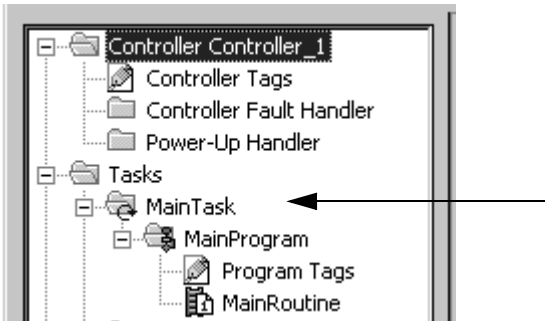
To choose how to configure output processing for a task:

Figure 3.1 Choose how to configure output processing for a task.

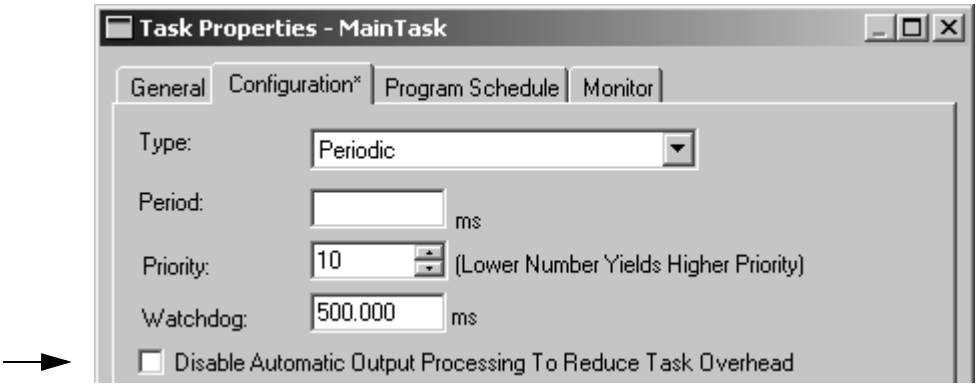


Manually Configure Output Processing

1. In the controller organizer, right-click the task and select Properties.



2. Select the Configuration tab.



3. Configure output processing for the task:

If you want to	Then
enable the processing of outputs at the end of the task	Clear (uncheck) the Disable Automatic Output Processing To Reduce Task Overhead check box (default).
disable the processing of outputs at the end of the task	Select (check) the Disable Automatic Output Processing To Reduce Task Overhead check box.

4. Click OK.

Programmatically Configure Output Processing

To write logic to configure output processing for a task, use a Set System Value (SSV) instruction. Access the attribute of the TASK object for the task:

If you want to	Then access this attribute	Data Type	Instruction	Description	
enable or disable the processing of outputs at the end of a task	DisableUpdateOutputs	DINT	GSV	**To:**	**Set the attribute to:**
			SSV	enable the processing of outputs at the end of the task	0
				disable the processing of outputs at the end of the task	1 (or any non-zero value)

EXAMPLE Programmatically Configure Output Processing

If Condition_1 = 0 then let Task_2 process outputs when it is done.

1. The ONS instruction limits the true execution of the SSV instruction to one scan.
2. The SSV instruction sets the DisableUpdateOutputs attribute of Task_2 = 0. This lets the task automatically process outputs when it finishes its execution.



If Condition_1 = 1 then do not let Task_2 process outputs when it is done.

1. The ONS instruction limits the true execution of the SSV instruction to one scan.
2. The SSV instruction sets the DisableUpdateOutputs attribute of Task_2 = 1. This prevents the task from automatically processing outputs when it finishes its execution.



Inhibit a Task

By default, each task executes based on its trigger (event, periodic, or continuous). As an option, you can prevent a task from executing when its trigger occurs (i.e., inhibit the task). This is useful to test, diagnose, or start up your project.

If you want to	Then
let the task execute when its trigger occurs	Uninhibit the task (default).
prevent the task from executing when its trigger occurs	Inhibit the task.

EXAMPLE *Inhibit a Task*

During the commissioning of a system that uses several task, you can first test each task individually.

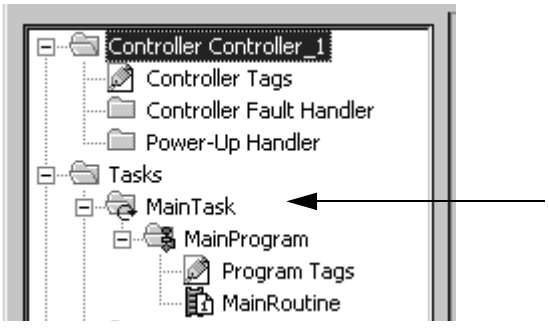
1. Inhibit all the tasks except one, and then test that task.
2. Once the task meets your requirements, inhibit it and uninhibit a different task.
3. Continue this process until you have tested all your tasks.

If a task is inhibited, the controller still prescans the task when the controller transitions from program to run or test mode.

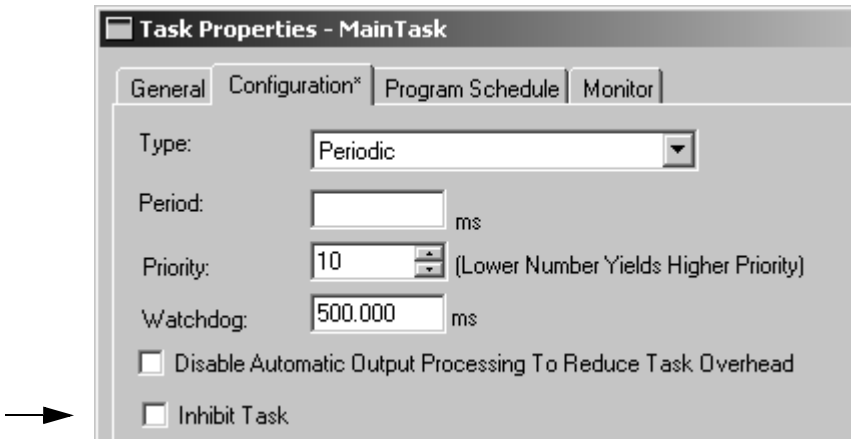
Manually Inhibit or Uninhibit a Task

To manually inhibit or uninhibit the execution of a task:

1. In the controller organizer, right-click the task and select Properties.



2. Select the Configuration tab.



3. Inhibit or uninhibit the task:

If you want to	Then
let the task execute when its trigger occurs	Uncheck the Inhibit Task check box (default).
prevent the task from executing when its trigger occurs	Check the Inhibit Task check box.

4. Click OK.

Programmatically Inhibit or Uninhibit a Task

To write logic to inhibit or uninhibit a task, use a Set System Value (SSV) instruction to access the attribute of the TASK object for the task:

Attribute	Data Type	Instruction	Description	
InhibitTask	DINT	GSV	Prevents the task from executing.	
		SSV	**To**	**Set the attribute to**
			enable the task	0 (default)
			inhibit (disable) the task	1 (or any non-zero value)

EXAMPLE Programmatically Inhibit or Uninhibit a Task

If Condition_1 = 0 then let Task_2 execute.

1. The ONS instruction limits the true execution of the SSV instruction to one scan.
2. The SSV instruction sets the InhibitTask attribute of Task_2 = 0. This uninhibits the task.



If Condition_1 = 1 then do not let Task_2 execute.

1. The ONS instruction limits the true execution of the SSV instruction to one scan.
2. The SSV instruction sets the InhibitTask attribute of Task_2 = 1. This inhibits the task.



Choose the Trigger for an Event Task

If configured correctly, an event task interrupts all other tasks for the minimum amount of time required to respond to the event. Each event task requires a specific trigger that defines when the task is to execute.

To trigger an event task when	Use this trigger	With these considerations
Digital input turns on or off	Module Input Data State Change	• Only one input module can trigger a specific event task. • The input module triggers the event task based on the change of state (COS) configuration for the module. The COS configuration defines which points prompt the module to produce data if they turn on or off. This production of data (due to COS) triggers the event task. • Typically, enable COS for only one point on the module. If you enable COS for multiple points, a task overlap of the event task may occur.
Analog module samples data	Module Input Data State Change	• Only one input module can trigger a specific event task. • The analog module triggers the event task after each real time sample (RTS) of the channels. • All the channels of the module use the same RTS.
Controller gets new data via a consumed tag	Consumed Tag	• Only one consumed can trigger a specific event task. • Typically, use an IOT instruction in the producing controller to signal the production of new data. The IOT instruction sets an event trigger in the producing tag. This trigger passes to the consumed tag and triggers the event task. • When a consumed tag triggers an event task, the event task waits for all the data to arrive before the event task executes.
Registration input for an axis turns on (or off)	Axis Registration 1 or 2	• In order for the registration input to trigger the event task, first execute a Motion Arm Registration (MAR) instruction. This lets the axis detect the registration input and in turn trigger the event task. • Once the registration input triggers the event task, execute the MAR instruction again to re-arm the axis for the next registration input. • If the scan time of your normal logic is *not* fast enough to re-arm the axis for the next registration input, consider placing the MAR instruction within the event task.
Axis reaches the position that is defined as the watch point	Axis Watch	• In order for the registration input to trigger the event task, first execute a Motion Arm Watch (MAW) instruction. This lets the axis detect the watch position and in turn trigger the event task. • Once the watch position triggers the event task, execute the MAW instruction again to re-arm the axis for the next watch position. • If the scan time of your normal logic is *not* fast enough to re-arm the axis for the next watch position, consider placing the MAW instruction within the event task
Motion planner completes its execution	Motion Group Execution	• The coarse update period for the motion group triggers the execution of both the motion planner and the event task. • Because the motion planner interrupts all other tasks, it executes first. If you assign the event task as the highest priority task, it executes after the motion planner.
Specific condition or conditions occur within the logic of a program	EVENT instruction	Multiple EVENT instructions can trigger the same task. This lets you execute a task from different programs.

Here are some example situations for event tasks and the corresponding triggers:

For this example situation:	Use an event task with this trigger:
A packaging line glues boxes closed. When a box arrives at the gluing position, the controller must immediately execute the gluing routine.	Module Input Data State Change
A production line uses a proximity sensor to detect the presence of a part. Because the proximity sensor is on for only a very short time (pulse), the continuous task might miss the off to on transition of the sensor.	Module Input Data State Change
In an engine test stand, you must capture and archive each sample of analog data.	Module Input Data State Change
Controller A produces an array of production data for Controller B. You want to make sure that Controller B doesn't use the values while Controller A is updating the array:	Consumed Tag
In a line that packages candy bars, you have to make sure that the perforation occurs in the correct location on each bar. Each time the registration sensor detects the registration mark, check the accuracy of an axis and perform any required adjustment.	Axis Registration 1 or 2
At the labeling station of a bottling line, you want to check the position of the label on the bottle. When the axis reaches the position that is defined as the watch point, check the label.	Axis Watch
A gluing station must adjust the amount of glue it applies to compensate for changes in the speed of the axis. After the motion planner executes, check the command speed of the axis and vary the amount of glue, if needed.	Motion Group Execution
In a production line, if any of the programs detect an unsafe condition the entire line must shut down. The shutdown procedure is the same regardless of the unsafe condition.	EVENT instruction

The triggers that you can use for an event task varies depending on your type of Logix5000 controller.

IMPORTANT RSLogix 5000 software may let you configure a trigger for an event task that your controller does not support. The project will verify and successfully download, but the event task will not execute.

Table 3.1 Determine which Logix5000 controllers support each type of event trigger.

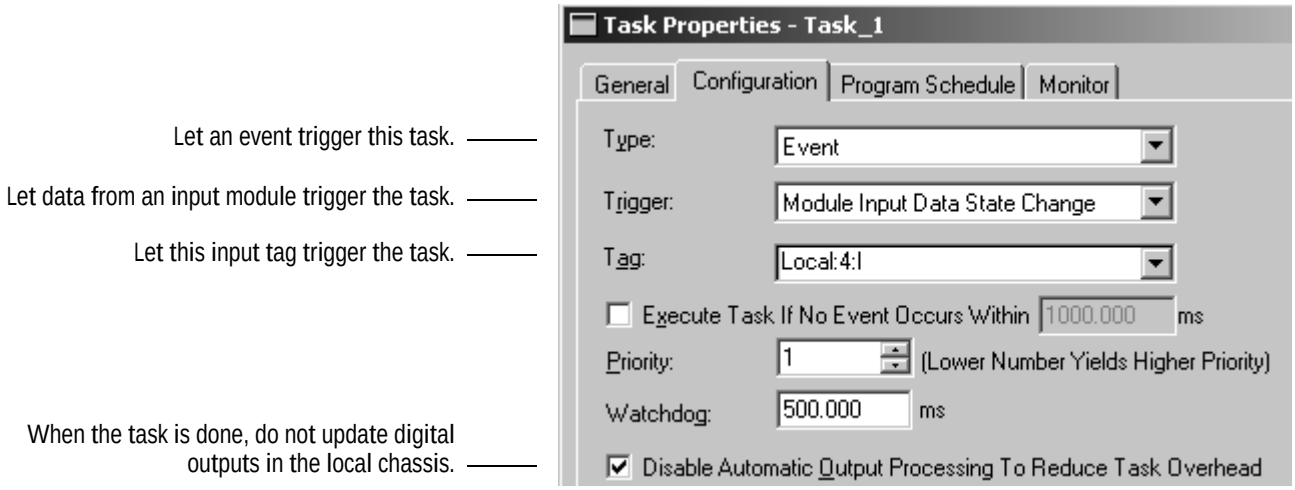
If you have this controller:	Then you can use these event task triggers:					
	Module Input Data State Change	Consumed Tag	Axis Registration 1 or 2	Axis Watch	Motion Group Execution	EVENT instruction
CompactLogix		4				4
FlexLogix		4				4
ControlLogix	4	4	4	4	4	4
DriveLogix		4	4	4	4	4
SoftLogix5800	4(1)	4(2)	4	4	4	4

(1) Requires a 1756 I/O module or a virtual backplane.

(2) A SoftLogix5800 controller produces and consumes tags only over a ControlNet network.

Using the Module Input Data State Change Trigger

To trigger an event task based on data from an input module, use the Module Input Data State Change trigger.



How an I/O Module Triggers an Event Task

These terms apply to the operation of an input module:

Term:	Definition:
multicast	A mechanism where a module sends data on a network that is simultaneously received by more that one listener (device). Describes the feature of the Logix5000 I/O line which supports multiple controllers receiving input data from the same I/O module at the same time.
requested packet interval (RPI)	The RPI specifies the interval at which a module multicasts its data. For example, an input module sends data to a controller at the RPI that you assign to the module. • The range is 0.2 ms (200 microseconds) to 750 ms. • When the specified time frame elapses, the module multicasts its data. This is also called a cyclic update.
real time sample (RTS)	The RTS specifies when an analog module scans its channels and multicasts the data (update the input data buffer then multicast). • The *RPI* specifies when the module multicasts the current contents of the input data buffer without scanning (updating) the channels. • The module resets the RPI timer each time and RTS transfer occurs.
change of state (COS)	The COS parameter instructs a digital input module to multicast data whenever a specified input point transitions from On → Off or Off → On. • You enable COS on a per-point basis. • When any point that is enabled for COS receives the specified change, the module multicasts the data for all its points. • By default, COS is enabled for both On → Off and Off → On changes for all points. • You must specify an RPI regardless of whether you enable COS. If a change does not occur within the RPI, the module sends its data at the RPI.

This table summarizes when an input module multicasts its data and triggers an event task within its own chassis.

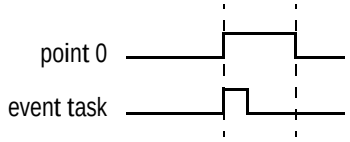
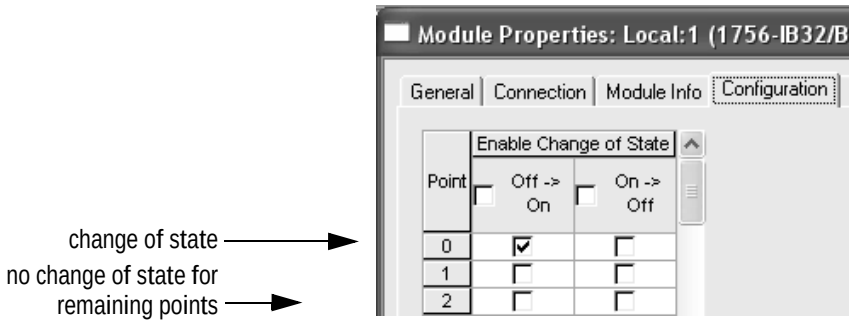
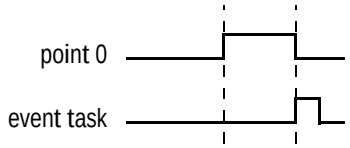
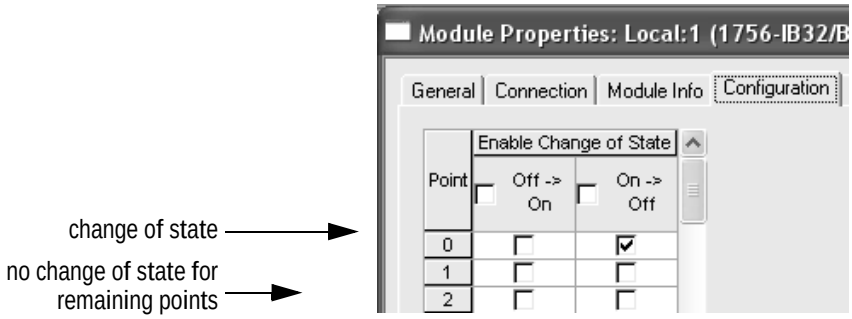
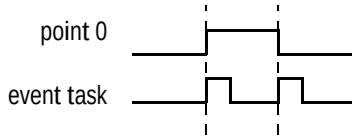
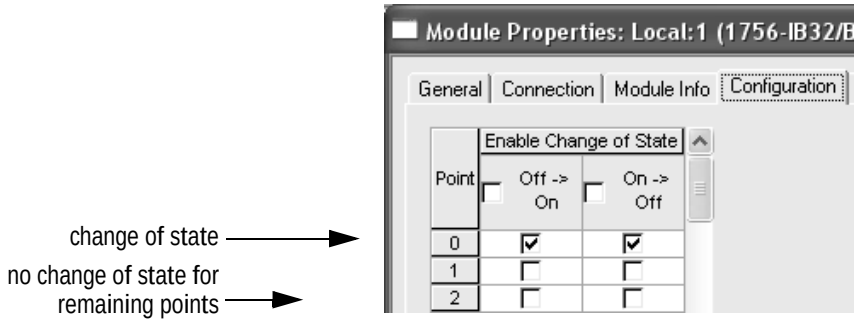
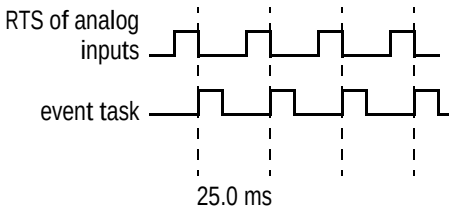
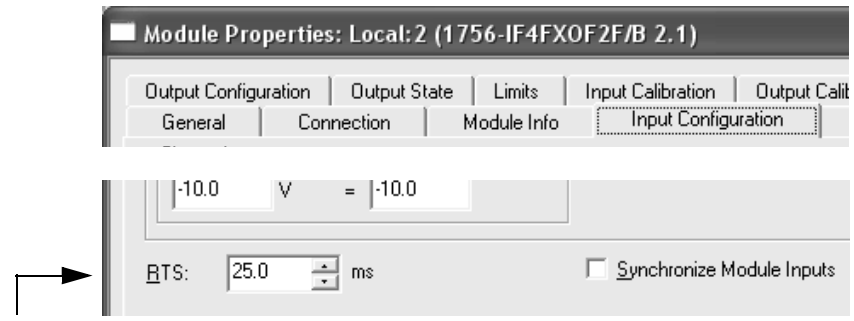
If the input module is:	And:	Then it multicasts data:	And it triggers an event task:
digital	COS is enabled for any point on the module	• when any point that is enabled for COS receives the specified change • at the RPI	when any point that is enabled for COS receives the specified change
	COS is not enabled for any point on the module	at the RPI	never
analog	RTS ≤ RPI	at the RTS (newly updated channel data)	at the RTS for the module
	RTS > RPI	• at the RTS (newly updated channel data) • at the RPI (does not contain updated data from the channels)	at the RTS for the module

If the module is in a remote chassis, only the RPI determines when the controller receives the data and event trigger over the network.

Over this network:	The controller receives the data:
EtherNet/IP	close to the RPI, on average
ControlNet	at the actual packet interval (≤ RPI)

Here are some examples that show COS and RTS configurations:

IMPORTANT If you use a digital module to trigger an event task, configure only one point on the module for COS. If you configure multiple points, a task overlap could occur.

If you want this:	Then configure the input module like this (point 0 is used as an example only):
	
	
	
	

Make Sure Your Module Can Trigger an Event Task

To use an input module to trigger an event task, the module must support event task triggering. If the module is in a remote location, the associated communication modules must also support event triggering.

This table lists Rockwell Automation modules that we have tested for event task triggering. Some 3rd party modules may also support event task triggering. Before you use a 3rd party module, check with the supplier to validate the operation of the module.

Category:	Modules:	
Digital I/O modules that support change of state	1756-IA8D 1756-IA16I 1756-IB16 1756-IB16I 1756-IB32 1756-IG16 1756-IH16ISOE 1756-IN16 1756-IV32	1756-IA16 1756-IA32 1756-IB16D 1756-IB16ISOE 1756-IC16 1756-IH16I 1756-IM16I 1756-IV16
Analog I/O modules that support real time sample	1756-IF16 1756-IF6CIS 1756-IF8 1756-IT6I	1756-IF4FXOF2F/A 1756-IF6I 1756-IR6I 1756-IT6I2
Communication modules that provide rack-optimized connections	1756-CNB/A 1756-CNB/D 1756-CNBR/B 1756-DNB 1756-SYNCH/A	1756-CNB/B 1756-CNBR/A 1756-CNBR/D 1756-ENBT/A 1784-PCIDS/A
Generic I/O modules that conform to CIP event communications	1756-MODULE 1789-MODULE	

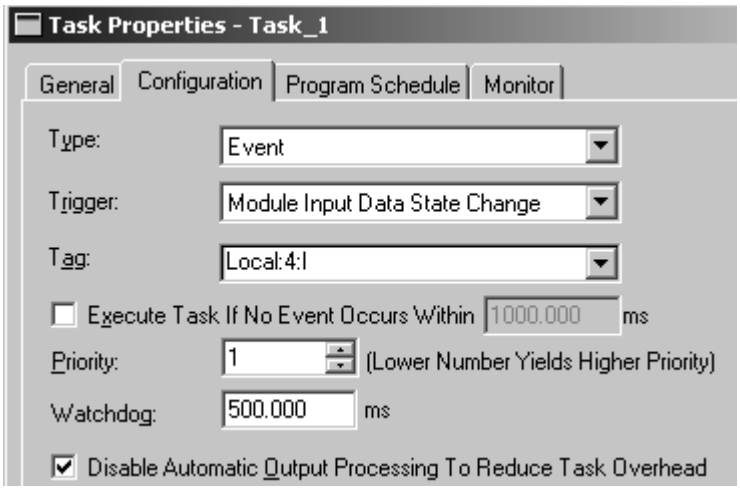
Checklist for an Input Event Task

For this:		Make sure you:
q	1. Input module type	For the fastest response, use these modules: • For fastest digital response, use a 1756-IB32/B module. • For fastest analog response, use a 1756-IF4FXOF2F module.
q	2. I/O module location	Place the module that triggers the event and the modules that respond to the event (outputs) in the same chassis as the controller. Remote modules add network communications to the response time.
q	3. Number of local modules	Limit the number of modules in the local chassis. Additional modules increases the potential for backplane delays
q	4. Change of state (COS)	If a digital device triggers the event, enable COS for only the point that triggers the event task. • Enable change of state for the type of transition that triggers the task, either Off → On, On → Off, or both. • If you configure COS for both Off → On and On → Off, the point triggers an event task whenever the point turns on or off. Make sure the duration of the input is longer than the scan time of the task. Otherwise an overlap could occur. • Disable (clear) COS for the remaining points on the input module. If you configure multiple points on a module for COS, each point could trigger the event task. This could cause an overlap.
q	5. Task priority	Configure the event task as the highest priority task. If a periodic task has a higher priority, the event task may have to wait until the periodic task is done.
q	6. Motion planner	The motion planner interrupts all other tasks, regardless of their priority. • The number of axes and coarse update period for the motion group effect how long and how often the motion planner executes. • If the motion planner is executing when a task is triggered, the task waits until the motion planner is done. • If the coarse update period occurs while a task is executing, the task pauses to let the motion planner execute.
q	7. Number of event tasks	Limit the number of event tasks. Each additional task reduces the processing time that is available for other tasks. This could cause an overlap.
q	8. Automatic Output Processing	For an event task, you can typically disable automatic output processing (default). This reduces the elapsed time of the task. To verify this decision, see Figure 3.1 on page 71.
q	9. IOT instruction	Use an IOT instruction for each output module that you reference in the event task. The IOT instruction overrides the RPI for the module and immediately sends the data.

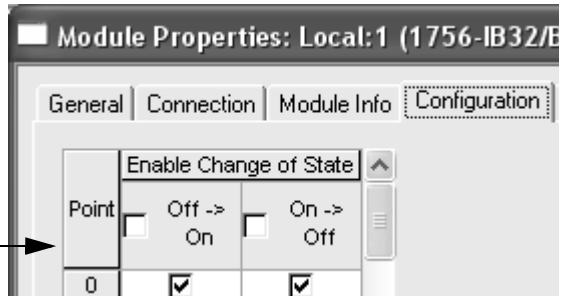
EXAMPLE

As parts move past a diverter location, the controller must decide whether or not to turn on the diverter. Once the diverter is on, the controller must also turn it off before the next part is in that position. Because of the speed of the line, an event task controls the diverter.

A photoeye at the diverter position indicates when a part is in the diverter position. The input is wired to the module in slot 4 of the local chassis.



The diverter photoeye (point 0) is configured for change of state for both Off and On. This lets the photoeye trigger the event task when it turns on and when it turns off.



The event task uses this logic to control the diverter.

If Diverter_Photoeye = 1 (part is in the diverter position)

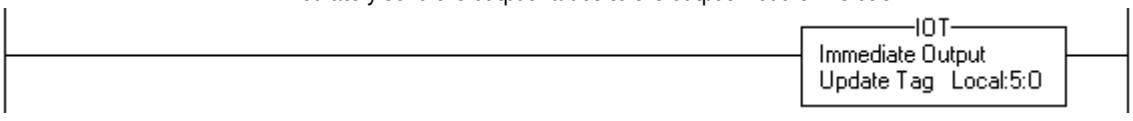
And Divert_Part = 1 (divert this part)

Then Diverter = 1 (turn on the diverter)

Otherwise Diverter = 0 (turn off the diverter)



Immediately send the output values to the output module in slot 5.



Estimate Throughput

To estimate the throughput time from input to output (screw to screw), use this worksheet:

Consideration:	Value:
1. What is the input filter time of the module that triggers the event task? This is typically shown in milliseconds. Convert it to microseconds (μs).	μs
2. What is the hardware response time for the input module that triggers the event task? Make sure you use the appropriate type of transition (Off → On or On → Off). See Table 3.2 on page 3-86.	μs
3. What is the backplane communication time?	
If the chassis size is: / **Use this value (worst case):** 4 slot / 13 μs 7 slot / 22 μs 10 slot / 32 μs 13 slot / 42 μs 17 slot / 54 μs	μs
4. What is the total execution time of the programs of the event task?	μs
5. What is the backplane communication time? (Same value as step 3.)	μs
6. What is the hardware response time of the output module.	μs
7. Add steps 1 through 6. This is the minimum estimated throughput, where execution of the motion planner or other tasks do *not* delay or interrupt the event task.	μs
8. What is the scan time of the motion group?	μs
9. What is the total scan time of the tasks that have a higher priority than this event task (if any)?	μs
10. Add steps 7 through 9. This is the nominal estimated throughput, where execution of the motion planner or other tasks delay or interrupt the event task.	μs

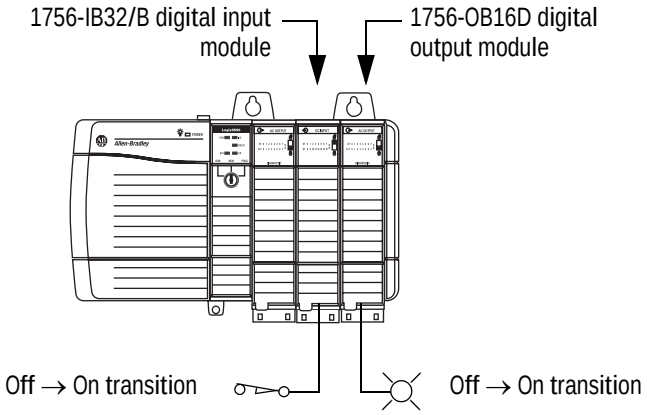
Table 3.2 Nominal hardware response times for the 1756 I/O modules most commonly used with Event tasks.

Module:	Nominal response time μs:			
	25° C		60° C	
	Off → On	On → Off	Off → On	On → Off
1756-IB16	265	582	265	638
1756-IB16D	303	613	305	673
1756-IB32/B	330	359	345	378
1756-IV16	257	435	254	489
1756-IV32	381	476	319	536
1756-OB16D	48	519	51	573
1756-OB16E	60	290	61	324
1756-OB32	38	160	49	179
1756-OV16E	67	260	65	326
1756-OV32E	65	174	66	210

EXAMPLE

Estimate Throughput

This example shows the throughput considerations for the system shown below. In this example, the throughput is the time from when the input turns on to when the output turns on.



Consideration:	Value:
1. What is the input filter time of the module that triggers the event task? This is typically shown in milliseconds. Convert it to microseconds (μs).	0 μs
2. What is the hardware response time for the input module that triggers the event task? Make sure you use the appropriate type of transition (Off → On or On → Off). See Table 3.2 on page 3-86.	330 μs
3. What is the backplane communication time?	13 μs
4. What is the total execution time of the programs of the event task?	400 μs
5. What is the backplane communication time? (Same value as step 3.)	13 μs
6. What is the hardware response time of the output module.	51 μs
7. Add steps 1 through 6. This is the minimum estimated throughput, where execution of the motion planner or other tasks do *not* delay or interrupt the event task.	807 μs
8. What is the scan time of the motion group?	1130 μs
9. What is the total scan time of the tasks that have a higher priority than this event task (if any)?	0 μs
10. Add steps 7 through 9. This is the nominal estimated throughput, where execution of the motion planner or other tasks delay or interrupt the event task.	1937 μs

If the chassis size is:	Use this value (worst case):
4 slot	13 μs
7 slot	22 μs
10 slot	32 μs
13 slot	42 μs
17 slot	54 μs

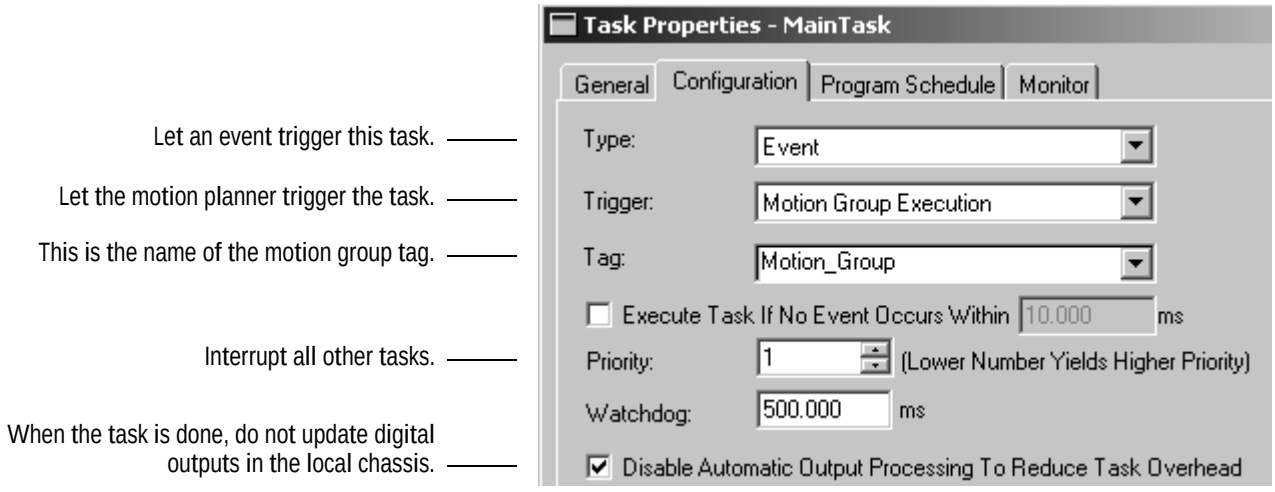
Additional Considerations

These considerations effect the scan time of the event task, which effects the speed at which it can respond to the input signal.

Consideration:	Description:
amount of code in the event task	Each logic element (rung, instruction, Structured Text construct, etc…) adds scan time to the task.
task priority	If the event task is not the highest priority task, a higher priority task may delay or interrupt the execution of the event task.
CPS and UID instructions	If one of these instructions are active, the event task cannot interrupt the currently executing task. (The task with the CPS or UID.)
communication interrupts	These actions of the controller interrupt a task, regardless of the priority of the task • communication with I/O modules Modules that have large data packets have a greater impact, such as the 1756-DNB module. • serial port communication

Using the Motion Group Trigger

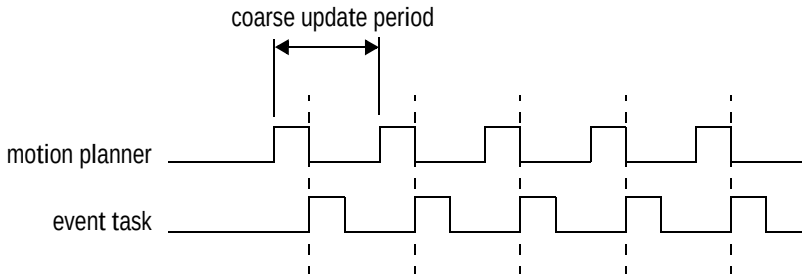
To couple the execution of an event task with the execution of the motion planner, use the Motion Group Execution trigger.



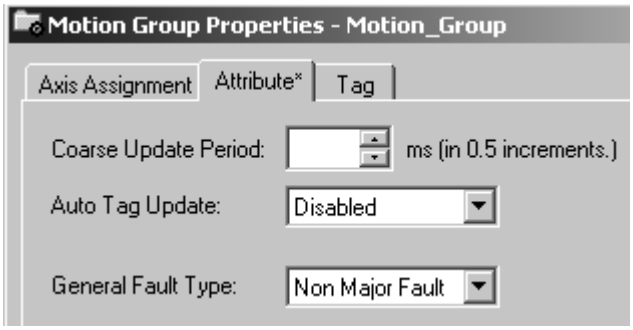
The Motion Group Execution trigger works as follows:

- The coarse update period for the motion group triggers the execution of both the motion planner and the event task.
- Because the motion planner interrupts all other tasks, it executes first. If you assign the event task as the highest priority task, it executes immediately after the motion planner.

This timing diagram shows the relationship between the motion planner and the event task.



The coarse update period for the motion group triggers both the motion planner and the event task.

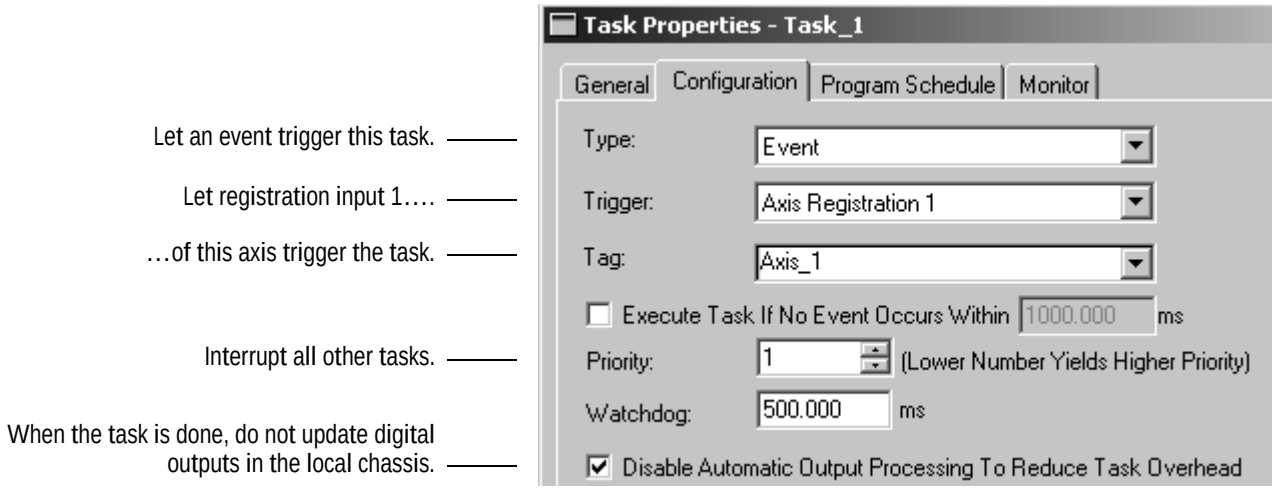


Checklist for a Motion Group Task

For this:		Make sure you:
q	1. Scan time	Make sure the scan time of the event task is significantly less than the course update period of the motion group. Otherwise, a task overlap could occur.
q	2. Task priority	Configure the event task as the highest priority task. If a periodic task has a higher priority, the event task may have to wait until the periodic task is done.
q	3. Number of event tasks	Limit the number of event tasks. Each additional task reduces the processing time that is available for other tasks. This could cause an overlap.
q	4. Automatic Output Processing	For an event task, you can typically disable automatic output processing (default). This reduces the elapsed time of the task. To verify this decision, see Figure 3.1 on page 71.

Using the Axis Registration Trigger

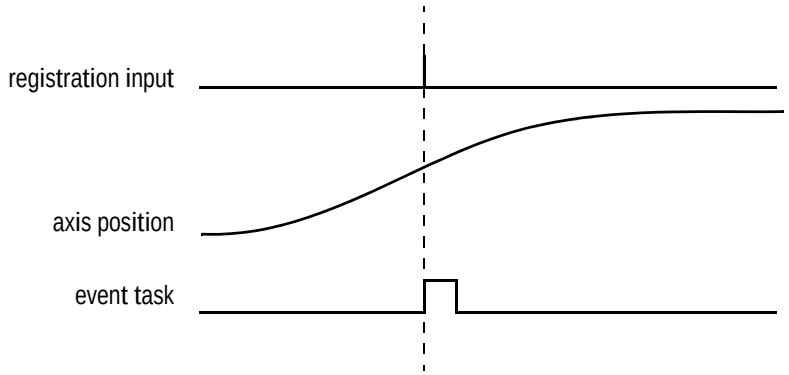
To let the registration input of an axis trigger an event task, use the Axis Registration (1 or 2) trigger.



When the specified registration input reaches its trigger condition, it triggers the event task.

- In the configuration of the event task, specify which registration input you want to trigger the task. Choose either Axis Registration 1 or Axis Registration 2.
- You must first arm the registration input using a Motion Arm Registration (MAR) instruction.
- In the MAR instruction, the Trigger Condition operand defines which transition of the registration input (Off → On or On → Off) triggers the event task.
- Once the registration input triggers the task, you have to re-arm the registration input.

This timing diagram shows the relationship between the registration input and the event task.



Checklist for an Axis Registration Task

For this:		Make sure you:	
q	1. Registration input	Arm the registration input (MAR instruction). This lets the axis detect the registration input and trigger the event task. • Initially, arm the registration input to detect the first trigger condition. • Re-arm the registration input after each execution of the event task. • Re-arm the registration input fast enough to detect each trigger condition.	
		If your normal logic is:	**Then:**
		fast enough to re-arm the registration input between intervals of the trigger condition E.g., Your normal logic always completes at least 2 scans between registration inputs.	Arm the registration input within your normal logic, if desired.
		not fast enough to re-arm the registration input	Arm the registration input within the event task.
q	2. Task priority	Configure the event task as the highest priority task. If a periodic task has a higher priority, the event task may have to wait until the periodic task is done.	
q	3. Number of event tasks	Limit the number of event tasks. Each additional task reduces the processing time that is available for other tasks. This could cause an overlap.	
q	4. Automatic Output Processing	For an event task, you can typically disable automatic output processing (default). This reduces the elapsed time of the task. To verify this decision, see Figure 3.1 on page 71.	

EXAMPLE

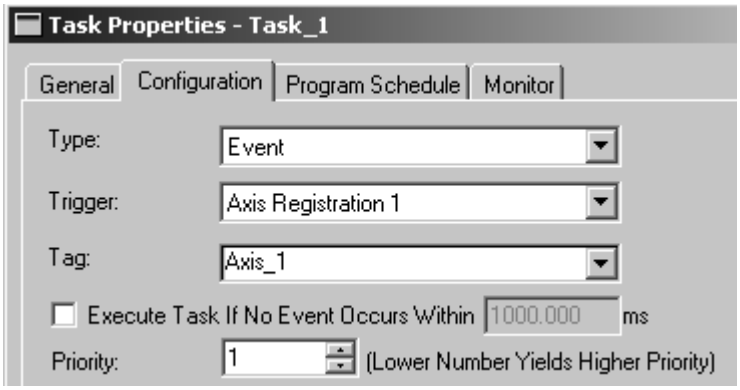
In a line that packages candy bars, you have to make sure that the perforation occurs in the correct location on each bar.

- Each time the registration sensor detects the registration mark, check the accuracy of an axis and perform any required adjustment.
- Due to the speed of the line, you have to arm the registration input within the event task.

A registration sensor is wired as registration input 1…

…for the axis named *Axis_1*.

This event task interrupts all other tasks.



This logic arms and re-arms the registration input.

Continuous task

If Arm_Registration = 1 (system is ready to look for the registration mark) then

The ONS instruction limits the execution of the EVENT instruction to one scan.

The EVENT instruction triggers an execution of Task_1 (event task).



Task_1 (event task)

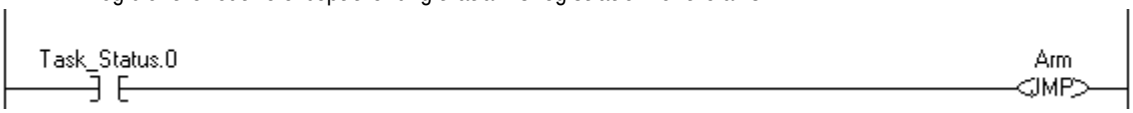
The GSV instruction sets Task_Status (DINT tag) = Status attribute for the event task. In the Instance Name attribute, THIS means the TASK object for the task that the instruction is in (i.e., Task_1).



continued on next page

If Task_Status.0 = 1 then an EVENT instruction triggered the event task. In the continuous task, the EVENT executes to arm registration for the first time.

The JMP instruction causes the controller to jump its execution to the Arm LBL instruction. This skips all the logic of the routine except the rung that arms registration for the axis.



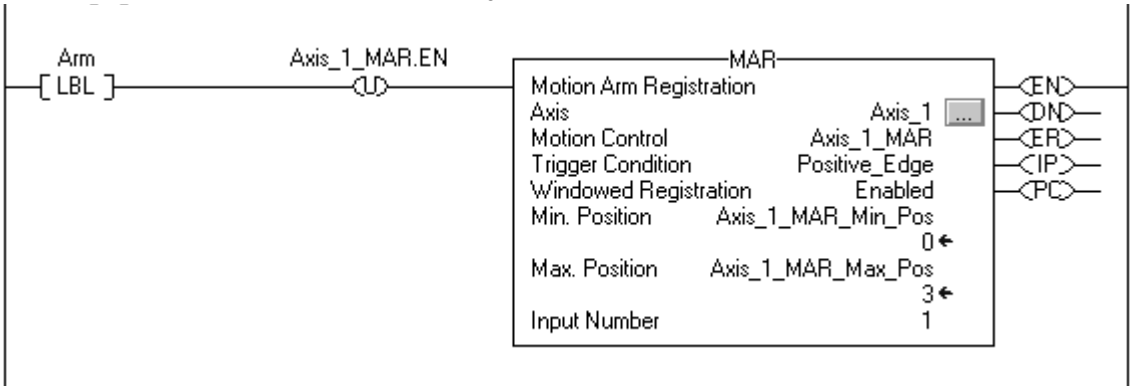
-
- Other logic
-

The MAR instruction executes each time the task executes and arms Axis_1 for registration.

The OTU instruction sets the EN bit of the MAR instruction = 0.

- The MAR instruction is a transitional instruction.
- To execute the MAR instruction, its rung-condition-in must go from false to true.
- By first clearing the EN bit, the instruction responds as if its rung-condition-in changed from false to true.

The MAR instruction arms the axis for registration.

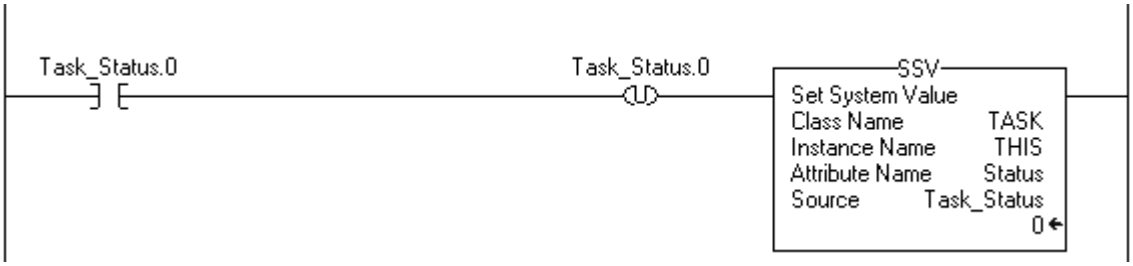


The controller does not clear the bits of the Status attribute once they are set. To use a bit for new status information, you must manually clear the bit.

If Task_Status.0 = 1 then clear that bit.

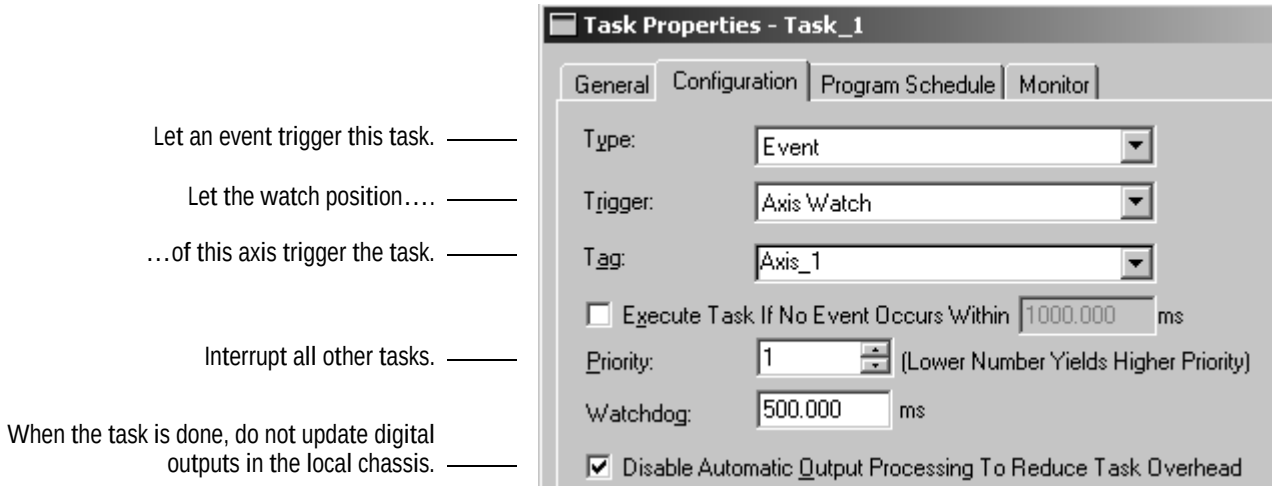
The OTU instruction sets Task_Status.0 = 0.

The SSV instruction sets the Status attribute of THIS task (Task_1) = Task_Status. This includes the cleared bit.



Using the Axis Watch Trigger

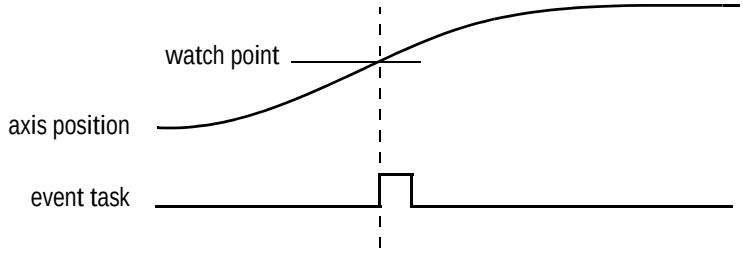
To let the watch position of an axis trigger an event task, use the Axis Watch trigger.



When the axis reaches the position that is specified as the watch position, it triggers the event task.

- You must first arm the axis for the watch position using a Motion Arm Watch (MAW) instruction.
- In the MAW instruction, the Trigger Condition operand defines the direction in which the axis must be moving to trigger the event task.
- Once the axis reaches the watch position and triggers the event task, you have to re-arm the axis for the next watch position.

This timing diagram shows the relationship between the watch position and the event task.



Checklist for an Axis Watch Task

For this:		Make sure you:	
q	1. Watch position	Use a MAW instruction to set up a watch position. This lets the axis trigger the event task when it reaches the watch position. • Initially, arm the axis to detect the first watch position. • Once the axis reaches the watch position and triggers the event task, re-arm the axis for the next watch position. • Re-arm the axis fast enough to detect each watch position.	
		If your normal logic is:	**Then:**
		fast enough to re-arm the axis between intervals of the watch position E.g., Your normal logic always completes at least 2 scans between watch positions.	Arm the axis within your normal logic, if desired.
		not fast enough to re-arm the axis	Arm the axis within the event task.
q	2. Task priority	Configure the event task as the highest priority task. If a periodic task has a higher priority, the event task may have to wait until the periodic task is done.	
q	3. Number of event tasks	Limit the number of event tasks. Each additional task reduces the processing time that is available for other tasks. This could cause an overlap.	
q	4. Automatic Output Processing	For an event task, you can typically disable automatic output processing (default). This reduces the elapsed time of the task. To verify this decision, see Figure 3.1 on page 71.	

EXAMPLE

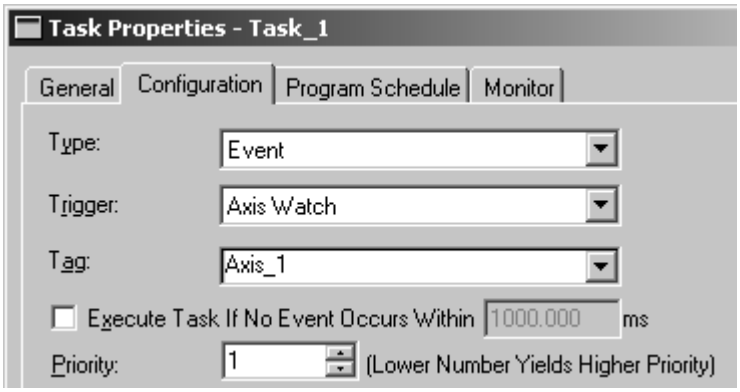
At the labeling station of a bottling line, you want to check the position of the label on the bottle.

- When the axis reaches the position that is defined as the watch point, check the label.and perform any required adjustment.
- Due to the speed of the line, you have to arm axis for the watch position within the event task.

Let the watch position…

…for the axis named *Axis_1* trigger the event task.

This event task interrupts all other tasks.



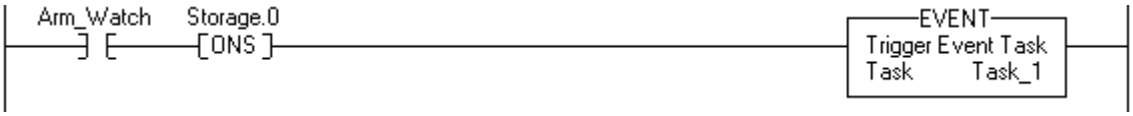
This logic arms and re-arms the axis for the watch position.

Continuous task

If Arm_Watch = 1 (system is ready to set up a watch position) then

The ONS instruction limits the execution of the EVENT instruction to one scan.

The EVENT instruction triggers an execution of Task_1 (event task).



Task_1 (event task)

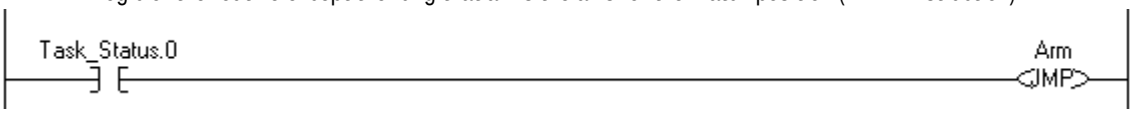
The GSV instruction sets Task_Status (DINT tag) = Status attribute for the event task. In the Instance Name attribute, THIS means the TASK object for the task that the instruction is in (i.e., Task_1).



continued on next page

If Task_Status.0 = 1 then an EVENT instruction triggered the event task. In the continuous task, the EVENT executes to set up the watch position for the first time.

The JMP instruction causes the controller to jump its execution to the Arm LBL instruction. This skips all the logic of the routine except the rung that arms the axis for the watch position (MAW instruction).



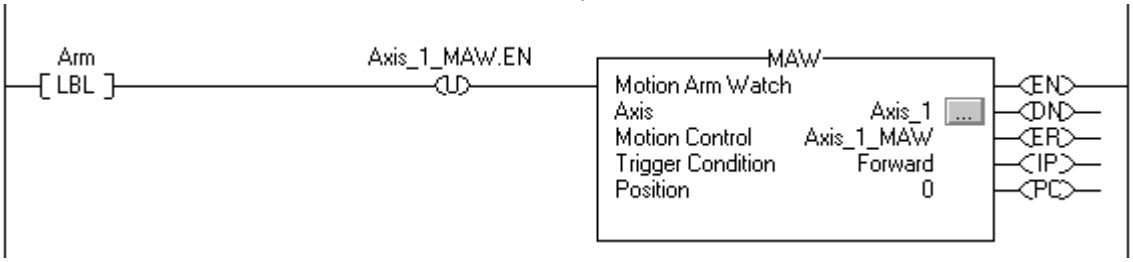
-
- Other logic
-

The MAW instruction executes each time the task executes and arms Axis_1 for the watch position.

The OTU instruction sets the EN bit of the MAW instruction = 0.

- The MAW instruction is a transitional instruction.
- To execute the MAW instruction, its rung-condition-in must go from false to true.
- By first clearing the EN bit, the instruction responds as if its rung-condition-in changed from false to true.

The MAW instruction arms the axis for the watch position.

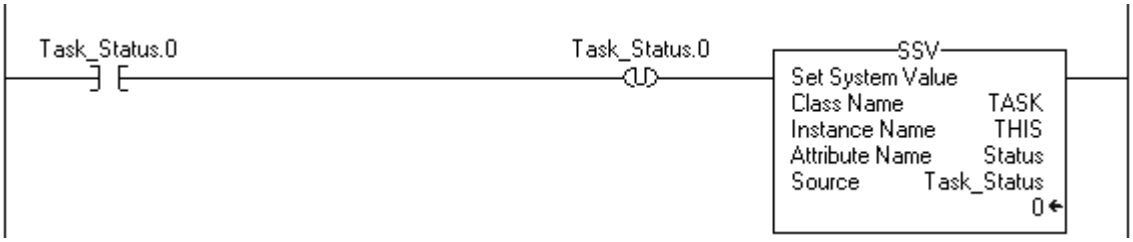


The controller does not clear the bits of the Status attribute once they are set. To use a bit for new status information, you must manually clear the bit.

If Task_Status.0 = 1 then clear that bit.

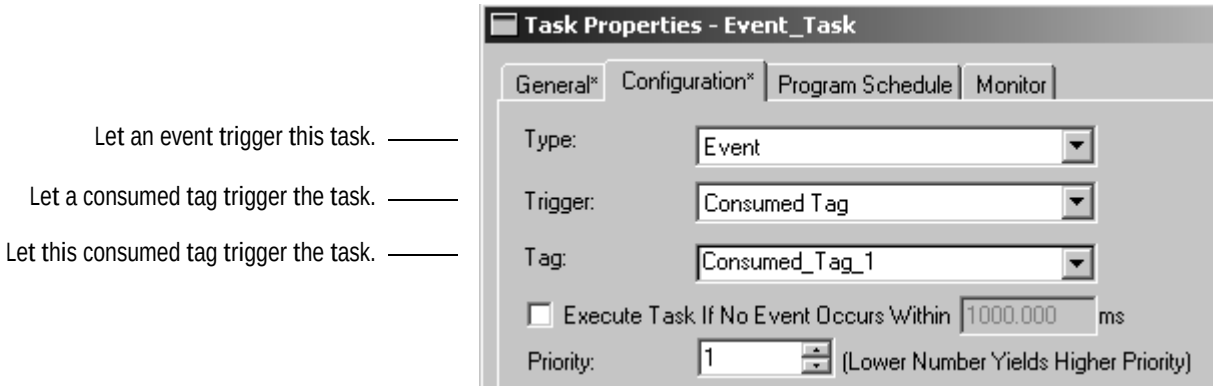
The OTU instruction sets Task_Status.0 = 0.

The SSV instruction sets the Status attribute of THIS task (Task_1) = Task_Status. This includes the cleared bit.

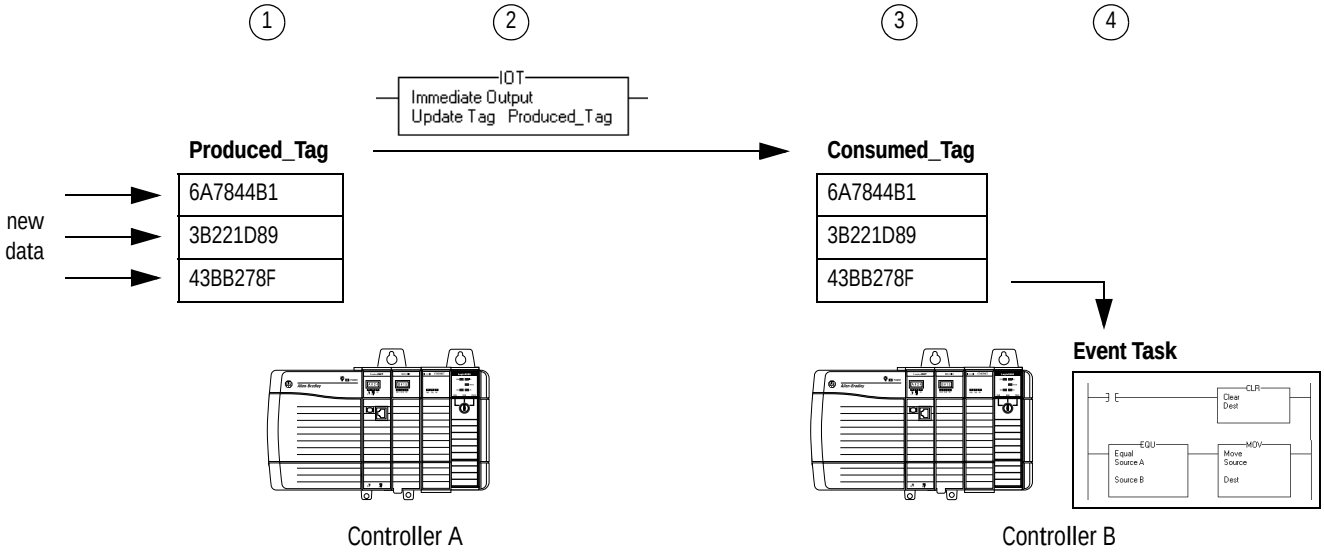


Using the Consumed Tag Trigger

To trigger an event task based on data from a consumed tag, use the Consumed Tag trigger.



A produced/consumed tag relationship can pass an event trigger along with data to a consumer controller. Typically, you use an Immediate Output (IOT) instruction to send the event trigger to the consumer controller.



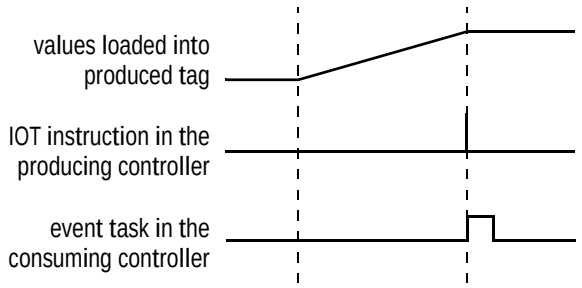
Description:	
1	In Controller A, logic updates the values of a produced tag.
2	Once the update is complete, the Controller A executes an IOT instruction to send the data and an event trigger to Controller B.
3	Controller B consumes the new data.
4	After Controller B updates the consumed tag, it executes the event task.

The type of network between the controllers determines when the consuming controller receives the new data and event trigger via the IOT instruction.

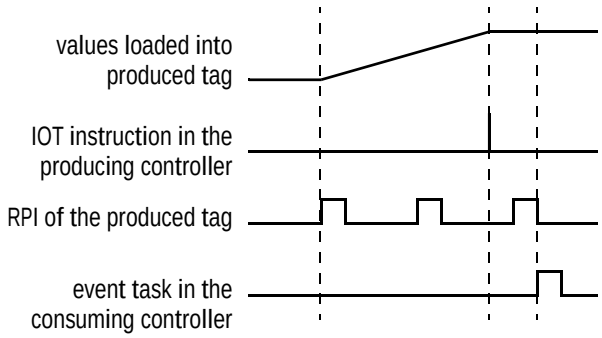
With this controller:	Over this network:	The consuming device receives the data and event trigger:
ControlLogix	backplane	immediately
	EtherNet/IP network	immediately
	ControlNet network	within the actual packet interval (API) of the consumed tag (connection)
SoftLogix5800	You can produce and consume tags only over a ControlNet network.	within the actual packet interval (API) of the consumed tag (connection)

These diagrams compare the receipt of data via an IOT instruction over EtherNet/IP and ControlNet networks.

EtherNet/IP network (ControlLogix controller)

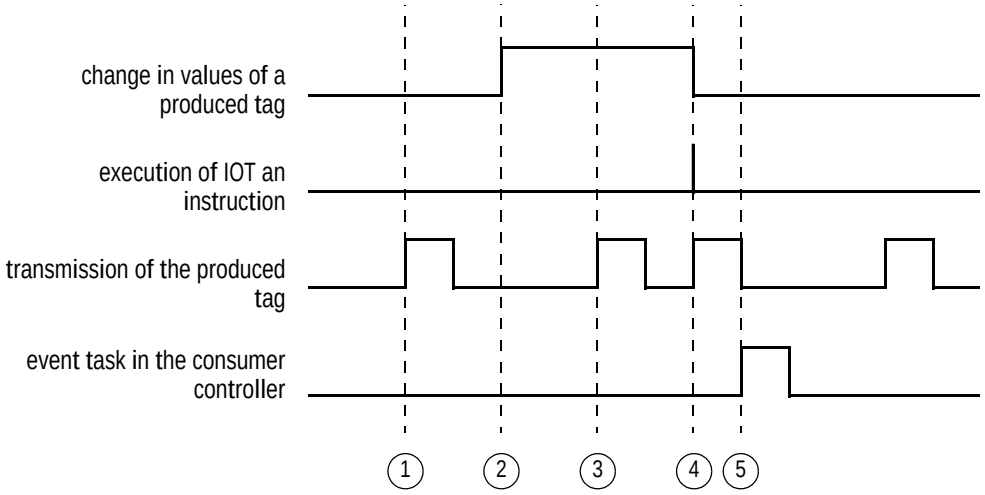


ControlNet network



Maintain the Integrity of Data

An event task with a consumed tag trigger provides a simple mechanism to pass data to a controller and ensure that the controller doesn't use the data while the data is changing.

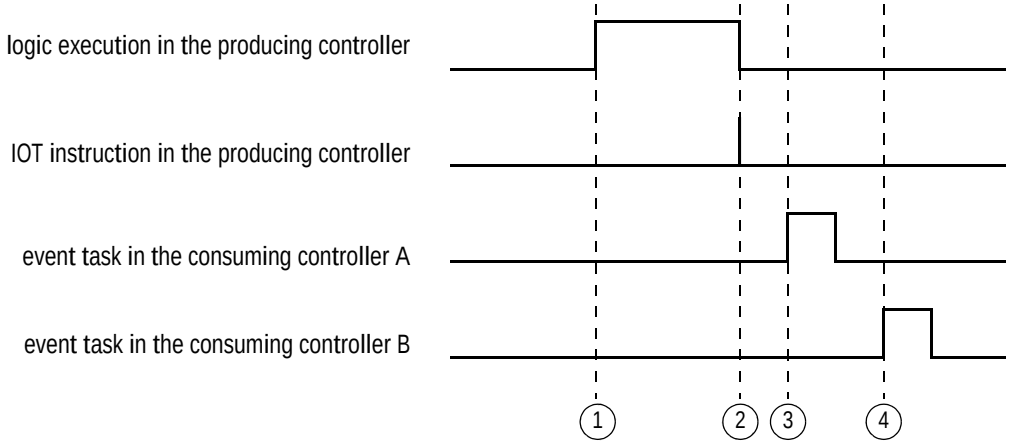


Description:	
1	RPI occurs for the produced tag. The produced tag transfers old data to the consuming controller.
2	The producer controller starts to update the values of the produced tag.
3	RPI occurs again for the produced tag. The produced tag transfers a mix of old and new data to the consuming controller.
4	The producer controller finishes updating the values of the produced tag. The producer controller executes an Immediate Output (IOT) instruction. The produced tag immediately transfers all the new data to the consuming controller.
5	When the consumer controller receives all the data, it executes its event task.

Although the producing controller executes the IOT instruction immediately after it loads new data, the event task is not triggered (in the consuming controller) until the consuming controller has received all the new data. This ensures that the controller operates on a complete packet of new data.

Synchronize Multiple Controllers

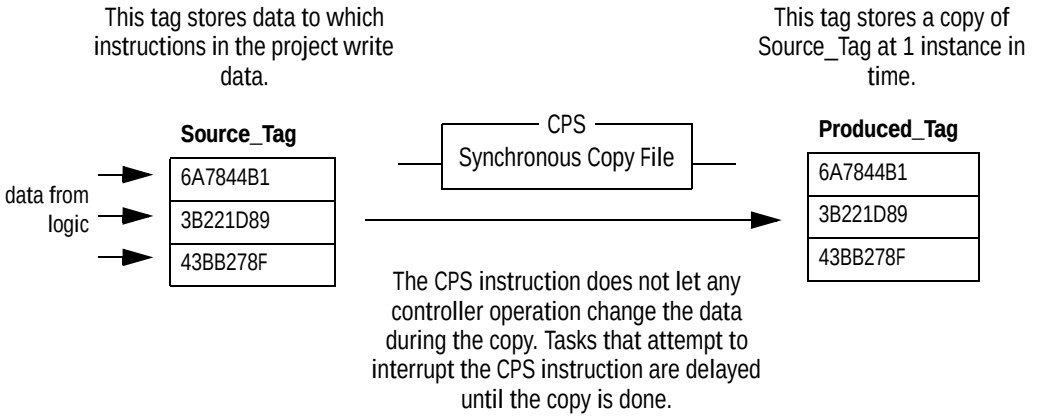
You can also use the produced/consumed tag relationship to synchronize controllers. In this case, the produced/consumed tag serves only as a triggering mechanism.



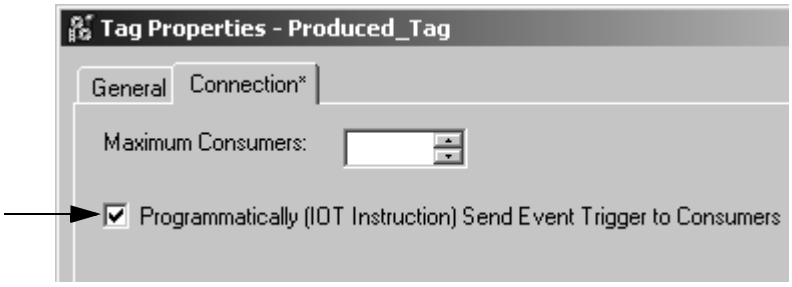
Description:	
1	The first controller executes an action with which other controllers need to stay synchronized.
2	When the action is done, the controller executes an IOT instruction. The IOT instruction uses a produced tag as its target.
3	When controller A receives the produced tag, it executes its event task.
4	When controller B receives the produced tag, it executes its event task.

Checklist for the Producer Controller

For this:		Make sure you:
q	1. Buffer of data	If you want to send a complete image of data at one instance in time, then produce a copy of the data, as shown below:



q	2. Produced tag properties	In the Connection properties of the produced tag, check this box:

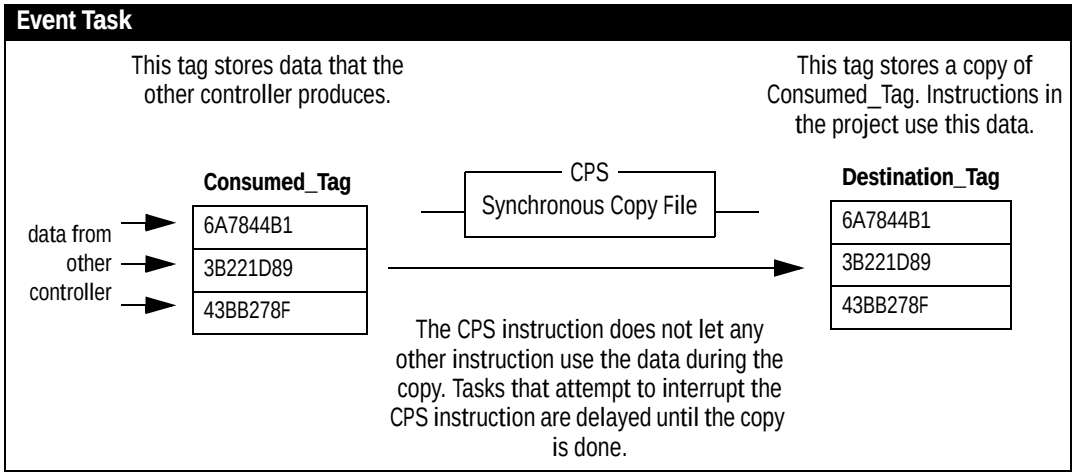


If you leave this box cleared (unchecked), the producing controller triggers the event task at the end of any task that automatically updates local outputs. In other words, the task scan will trigger the event in addition to the IOT instruction.

q	3. IOT instruction	Use an IOT instruction at the point in your logic where you want to trigger the event task. The IOT instruction overrides the RPI for the tag and immediately sends the event trigger and the data of the tag.

Checklist for the Consumer Controller

For this:		Make sure you:
q	1. Buffer of data	If you want to make sure that the controller does not use data from the consumed tag while the data is changing, use a copy of the consumed tag. Use the event task to copy the data, as shown below:



q	2. Task priority	Configure the event task as the highest priority task. If a periodic task has a higher priority, the event task may have to wait until the periodic task is done.
q	3. Number of event tasks	Limit the number of event tasks. Each additional task reduces the processing time that is available for other tasks. This could cause an overlap.
q	4. Automatic Output Processing	For an event task, you can typically disable automatic output processing (default). This reduces the elapsed time of the task. To verify this decision, see Figure 3.1 on page 71.

EXAMPLE

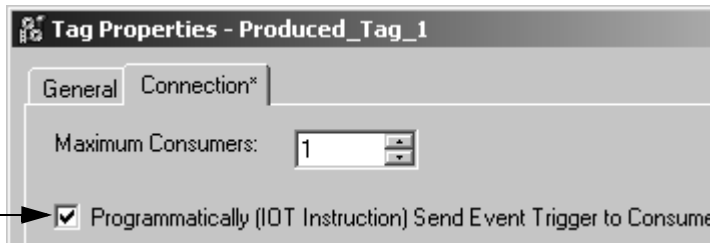
As parts move along a production line, each station requires production specifications for the part at its station. To make sure that a station doesn't act on old data, an event task signals the arrival of new data for the next part.

Producer Controller

This controller controls station 24 and produces data for the next station (station 25). To signal the transmission of new data, the controller uses these elements:

Produced Tag Properties

Produced_Tag is configured to update its event trigger via an IOT instruction.

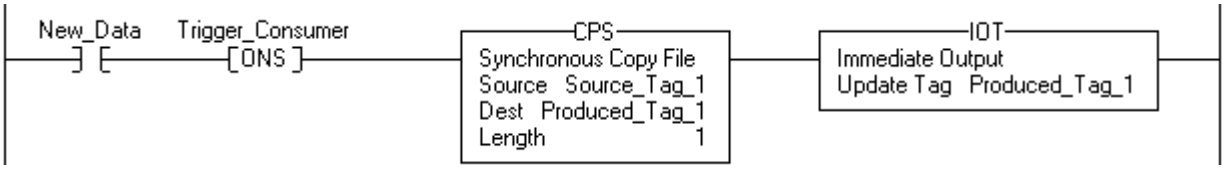


Ladder Logic

If New_Data = on, then this occurs for one scan:

The CPS instruction sets Produced_Tag_1 = Source_Tag_1.

The IOT instruction updates Produced_Tag_1 and sends this update to the consuming controller (station 25). When the consuming controller receives this update, it triggers the associated event task in that controller.



continued on next page

Consumer Controller

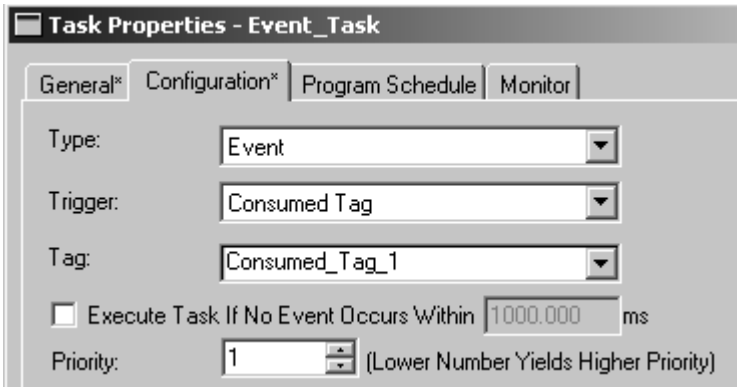
The controller at station 25 uses the data produced by station 24. To determine when new data has arrived, the controller uses an event task.

Event Task Properties

Let an event trigger this task. →

Let a consumed tag trigger the task. →

Let this consumed tag trigger the task. →



Ladder Diagram in the Event Task

When the event task executes, the CPS instruction sets Destination_Tag_1 = Consumed_Tag_1 (the values from the producing controller). The remaining logic in this controller uses the values from Destination_Tag_1.



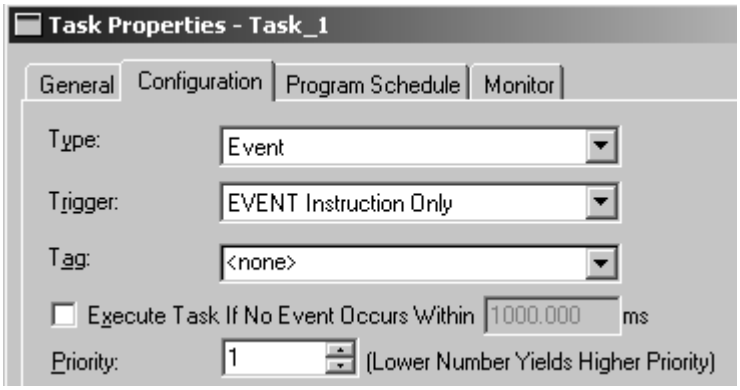
Using the EVENT Instruction Trigger

To trigger an event task based on conditions in your logic, use the EVENT Instruction Only trigger.

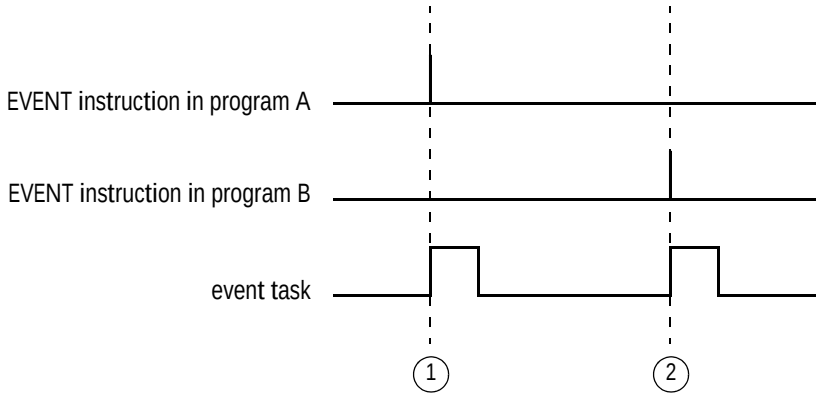
Let an event trigger this task.

Let an EVENT instruction trigger the task.

No tag is required.



The EVENT Instruction Only trigger requires that you use a Trigger Event Task (EVENT) instruction to trigger the task. You can use an EVENT instruction from multiple points in your project. Each time the instruction executes, it triggers the specified event task.



Description:	
1	Program A executes an EVENT instruction. The event task that is specified by the EVENT instruction executes one time.
2	Program B executes an EVENT instruction. The event task that is specified by the EVENT instruction executes one time.

Programmatically Determine if an EVENT Instruction Triggered a Task

To determine if an EVENT instruction triggered an event task, use a Get System Value (GSV) instruction to monitor the Status attribute of the task.

Table 3.3 Status Attribute of the TASK Object

Attribute:	Data Type:	Instruction:	Description:	
Status	DINT	GSV SSV	Provides status information about the task. Once the controller sets a bit, you must manually clear the bit to determine if another fault of that type occurred.	
			To determine if:	**Examine this bit:**
			An EVENT instruction triggered the task (event task only).	0
			A timeout triggered the task (event task only).	1
			An overlap occurred for this task.	2

The controller does not clear the bits of the Status attribute once they are set.

- To use a bit for new status information, you must manually clear the bit.
- Use a Set System Value (SSV) instruction to set the attribute to a different value.

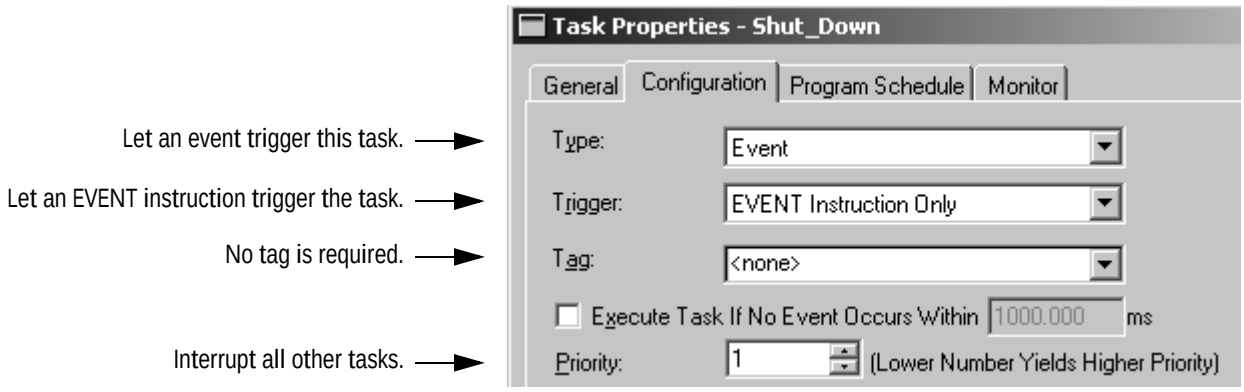
Checklist for an EVENT Instruction Task

For this:		Make sure you:
q	1. EVENT instruction	Use a Trigger Event Task (EVNT) instruction at each point in your logic that you want to trigger the event task.
q	2. Task priority	Configure the event task as the highest priority task. If a periodic task has a higher priority, the event task may have to wait until the periodic task is done.
q	3. Number of event tasks	Limit the number of event tasks. Each additional task reduces the processing time that is available for other tasks. This could cause an overlap.
q	4. Automatic Output Processing	For an event task, you can typically disable automatic output processing (default). This reduces the elapsed time of the task. To verify this decision, see Figure 3.1 on page 71.

EXAMPLE

A controller uses multiple programs but a common shut down procedure. Each program uses a program-scoped tag named Shut_Down_Line that turns on if the program detects a condition that requires a shut down.

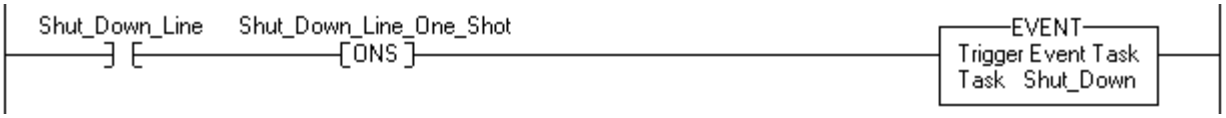
Event Task Properties



Ladder Diagram in Program_A

If Shut_Down_Line = on (conditions require a shut down) then

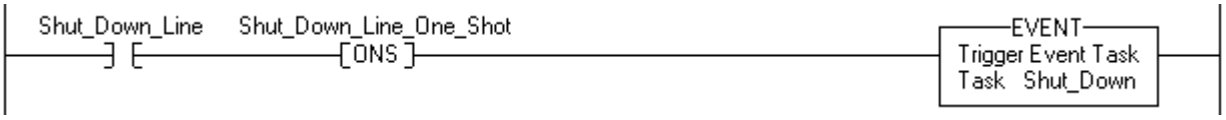
Execute the Shut_Down task one time



Ladder Diagram in Program_B

If Shut_Down_Line = on (conditions require a shut down) then

Execute the Shut_Down task one time

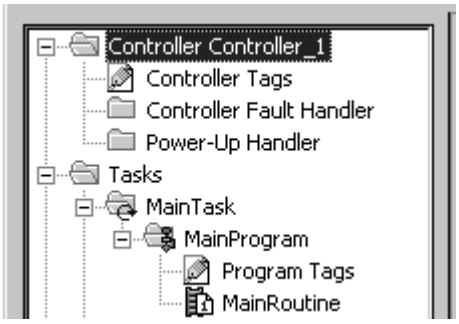


Create a Task

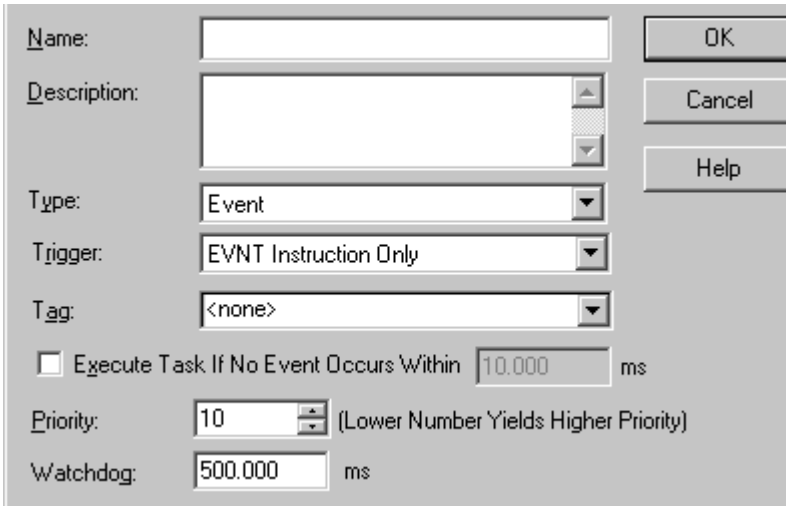
Create an Event Task

To create an event task:

1. In the controller organizer, right-click the Tasks folder and select New Task.



2. Type a **name** for the task. select Event for the task type, select a trigger for the task, select the tag for the triggering data, type the **priority** for the task, and type the **watchdog** time for the task.



3. Click OK

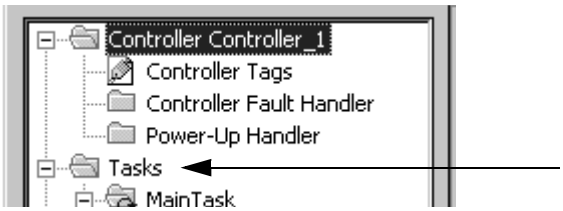
Create a Periodic Task

A periodic task performs a function or functions at a specific rate.

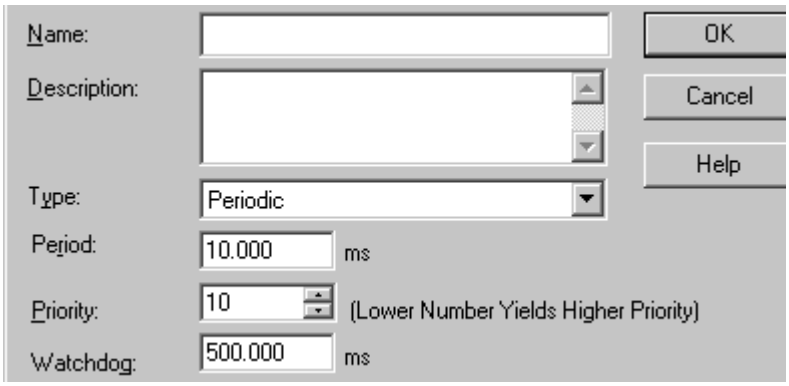
IMPORTANT Ensure that the time period is longer than the sum of the execution times of all the programs assigned to the task.

- If the controller detects that a periodic task trigger occurs for a task that is already operating, a minor fault occurs (overlap).
- Priorities and execution times of other tasks may also cause an overlap.

1. In the controller organizer, right-click the Tasks folder and select New Task.



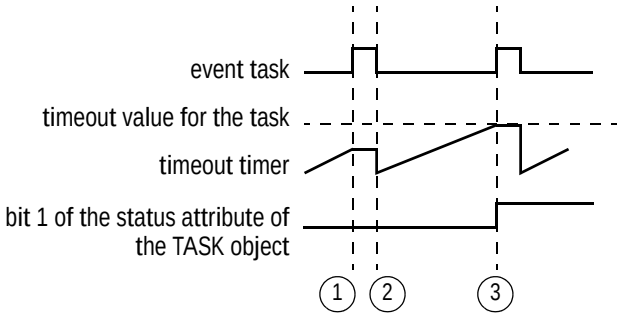
2. Type a **name** for the task, select Periodic (default) for the type of task, type the period at which you want the task to execute, type the **priority** for the task, and, type the **watchdog** time for the task.



3. Click 

Define a Timeout Value for an Event Task

If you want your event task to automatically execute if the trigger fails to occur within a certain time, assign a timeout value to the task. When the event task is done, its timeout timer begins to increment. If the timer reaches its preset value before the event task is triggered, the event task automatically executes.

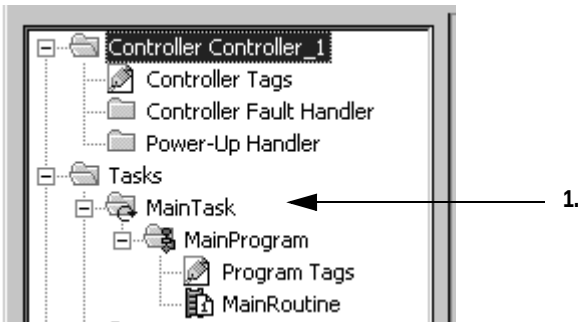


Description:	
1	Event task executes. Timeout time stops incrementing.
2	Event task is done. Timeout timer resets and begins incrementing.
3	Timeout timer reaches the timeout value. Event task automatically executes. In the Status attribute of the TASK object, bit 1 turns on.

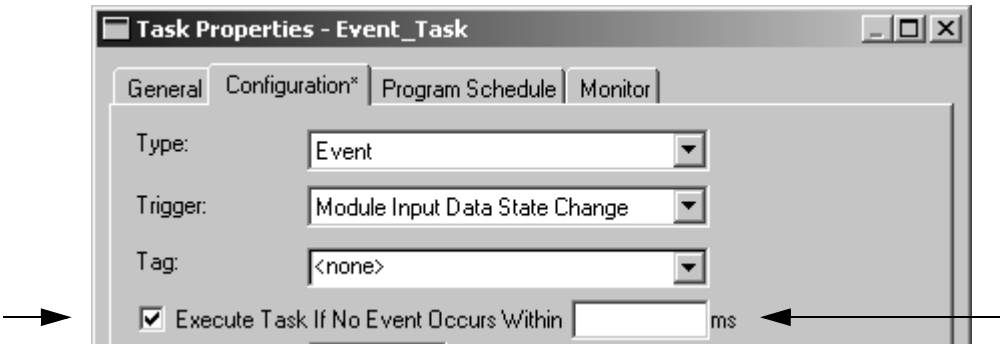
Assign a Timeout Value to an Event Task

To assign a timeout value to an event task:

1. In the controller organizer, right-click the event task and select Properties.



2. Select the Configuration tab.



3. Check the Execute Task If No Event Occurs Within check box and type the timeout value in milliseconds.

4. Click OK

Programmatically Configure a Timeout

To programmatically configure a timeout, use a Get System Value (GSV) instruction to access the attributes of the task.

Table 3.4 Status Attribute of the TASK Object

Attribute:	Data Type:	Instruction:	Description:	
Rate	DINT	GSV	**If the task type is:**	**Then the Rate attribute specifies the:**
		SSV	periodic	Period for the task. Time is in microseconds.
			event	The timeout value for the task. Time is in microseconds.
EnableTimeOut	DINT	GSV	Enables or disables the timeout function of an event task.	
		SSV	**To:**	**Set the attribute to:**
			disable the timeout function	0 (default)
			enable the timeout function	1 (or any non-zero value)

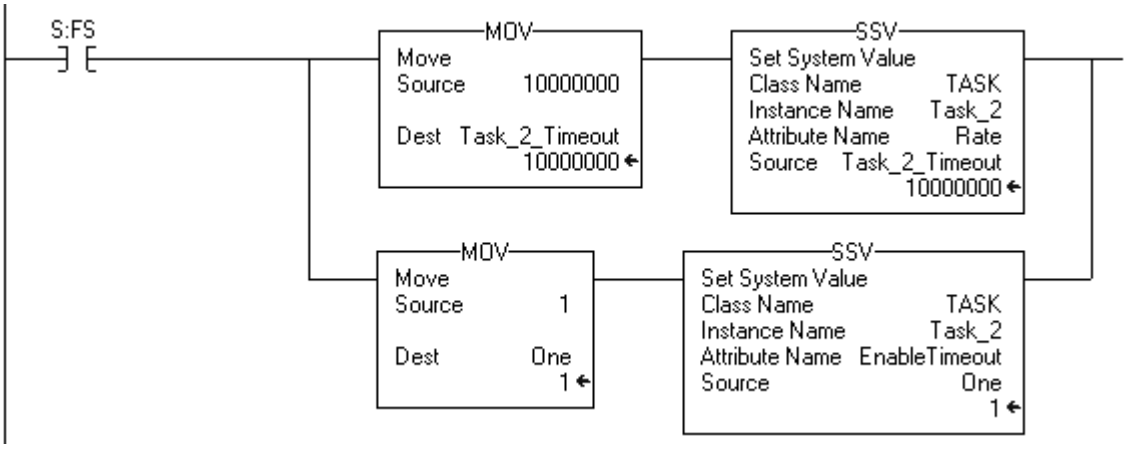
EXAMPLE

Programmatically Configure a Timeout

To make sure that a timeout value is always defined and enabled for an event task, the logic configures the timeout when the controller enters the run mode.

If S:FS = 1 (first scan) then set the timeout value for Task_2 and enable the timeout function:

1. The first MOV instruction sets Task_2_Timeout = 10000000 μs (DINT value). Then the SSV instruction sets the Rate attribute for Task_2 = Task_2_Timeout. This configures the timeout value for the task.
2. The second MOV instruction sets One = 1 (DINT value). Then the SSV instruction sets the EnableTimeout attribute for Task_2 = One. This enables the timeout function for the task.



Programmatically Determine if a Timeout Occurs

To determine if an event task executed due to a timeout, use a Get System Value (GSV) instruction to monitor the Status attribute of the task.

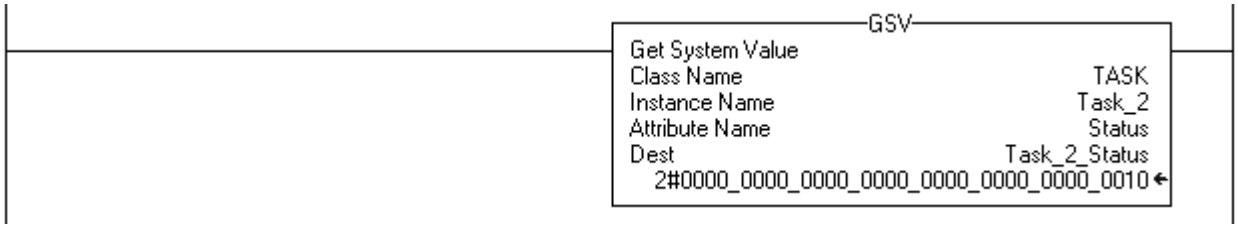
Table 3.5 Status Attribute of the TASK Object

Attribute:	Data Type:	Instruction:	Description:	
Status	DINT	GSV SSV	Provides status information about the task. Once the controller sets a bit, you must manually clear the bit to determine if another fault of that type occurred.	
			To determine if:	**Examine this bit:**
			An EVENT instruction triggered the task (event task only).	0
			A timeout triggered the task (event task only).	1
			An overlap occurred for this task.	2

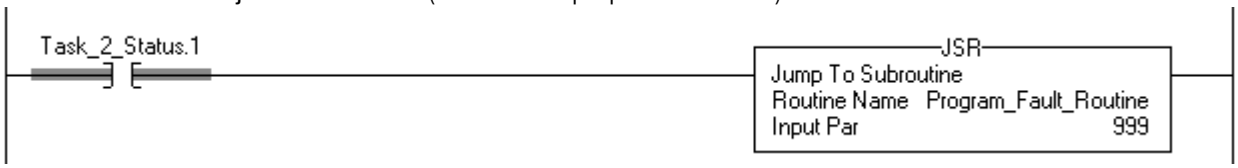
EXAMPLE Define a Timeout Value for an Event Task

If a timeout occurs for the event task, communication with the triggering device might have failed. This requires the process to shut down. To shut down the controller, the event task calls the fault routine for the program and supplies a user-defined fault code (999 in this example).

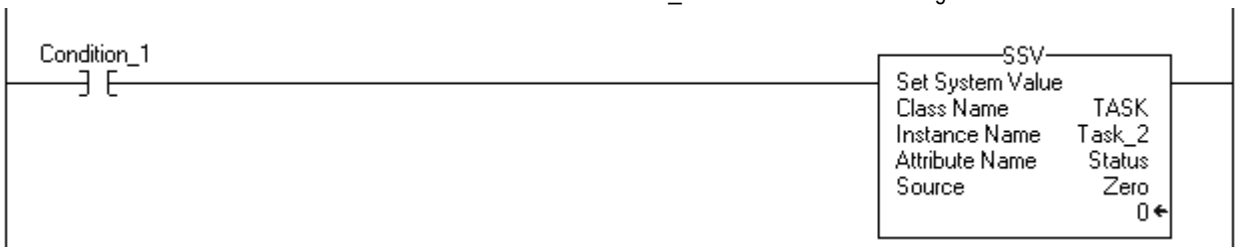
1. The GSV instruction sets Task_2_Status = Status attribute for Task_2 (DINT value).



2. If Task_2_Status.1 = 1, then a timeout occurred so shut down the controller and set the major fault code to 999:
   The JSR instruction calls the fault routine for the program. This produces a major fault.
   The major fault code = 999 (value of the input parameter of 999).



3. If Condition_1 = 1, then clear the bits of the Status attribute for Task_2:
   The SSV instruction sets the Status attribute of Task_2 = Zero. Zero is a DINT tag with a value of 0.



For more information on shutting down the controller, see "Create a User-Defined Major Fault"on page 409.

Adjust the System Overhead Time Slice

A Logix5000 controller communicates with a other devices (I/O modules, controllers, HMI terminals, etc.) at either a specified rate (scheduled) or when there is processing time available to service the communication (unscheduled).

This type of communication:	Is:
update I/O data (not including block-transfers)	Scheduled Communication
produce or consume tags	
communicate with programming devices (e.g., RSLogix 5000 software)	Unscheduled Communication
communicate with HMI devices	
execute Message (MSG) instructions, including block-transfers	
respond to messages from other controllers	
synchronize the secondary controller of a redundant system	
re-establish and monitor I/O connections (such as Removal and Insertion Under Power conditions); this *does not* include normal I/O updates that occur during the execution of logic.	
bridge communications from the serial port of the controller to other ControlLogix devices via the ControlLogix backplane	

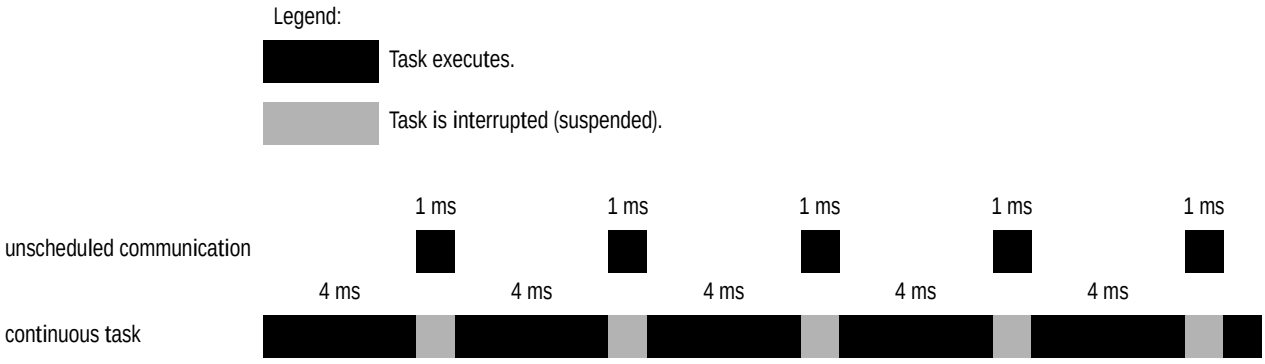
Unscheduled communication is any communication that you do not configure through the I/O configuration folder of the project.

- The **system overhead time slice** specifies the percentage of time (excluding the time for periodic or event tasks) that the controller devotes to unscheduled communication.
- The controller performs unscheduled communication for up to 1 ms at a time and then resumes the continuous task.

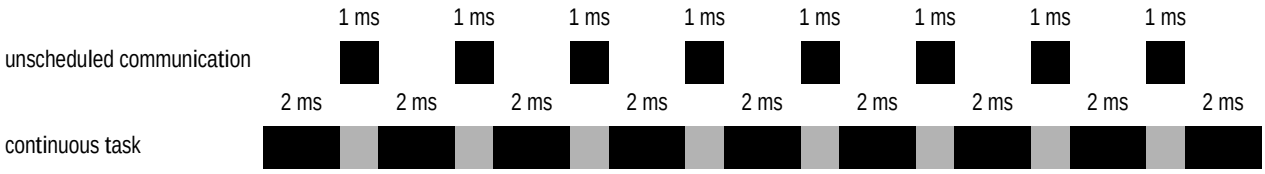
This table shows the ratio between the continuos task and unscheduled communication at various system overhead time slices:

At this time slice:	The continuous tasks runs for:	And unscheduled communication occurs for up to:
10%	9 ms	1 ms
20%	4 ms	1 ms
33%	2 ms	1 ms
50%	1 ms	1 ms

At a system overhead time slice to 20 %, unscheduled communication occurs every 4 ms of continuous task time for 1 ms.

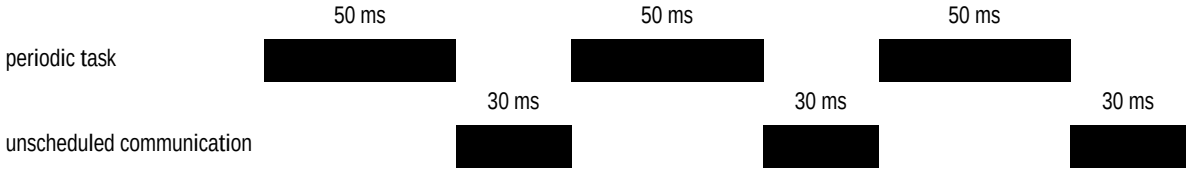


If you increase the system overhead time slice to 33 %, unscheduled communication occurs every 2 ms of continuous task time for 1 ms.



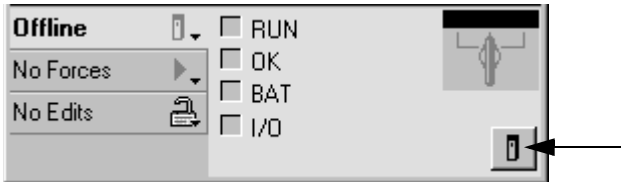
If the controller contains only a periodic task or tasks, the system overhead time slice value has no effect. Unscheduled communication occurs whenever a periodic task is not running.

For example, if your task takes 50 ms to execute and you configure its update rate to 80 ms, the controller has 30 ms out of every 80 ms for unscheduled communication.

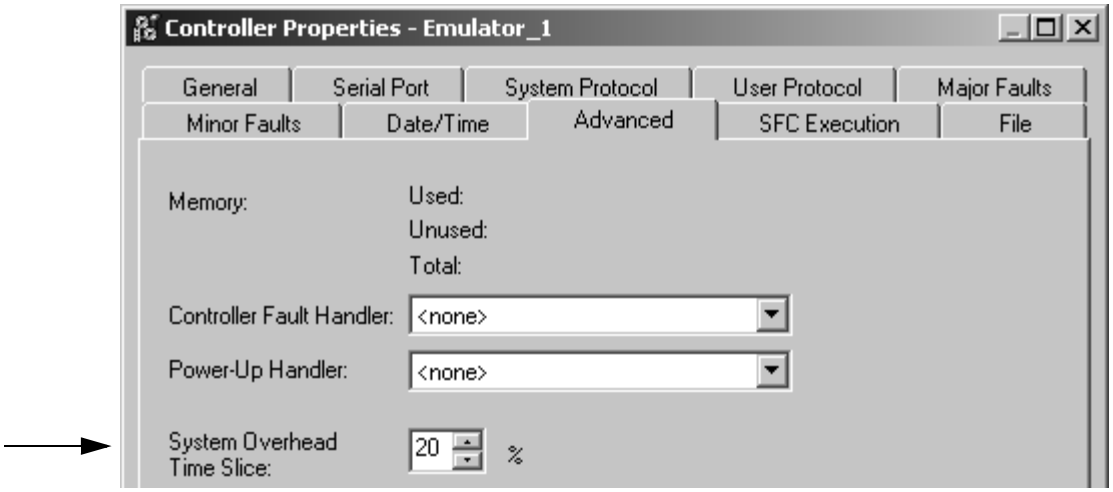


Adjust the System Overhead Time Slice

1. On the Online toolbar, click controller properties button.



2. Select the Advanced tab and type the system overhead timeslice.



3. Click OK

Adjust the Watchdog Time

Each task contains a watchdog timer that specifies how long a task can run before triggering a major fault.

ATTENTION



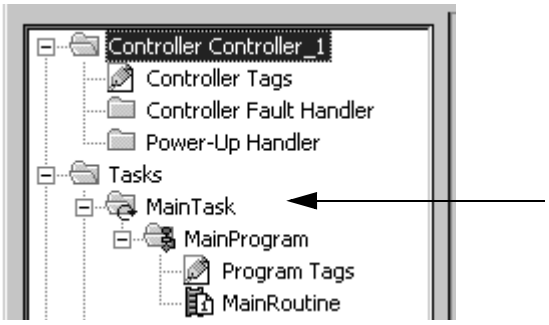
If the watchdog timer reaches a configurable preset, a major fault occurs. Depending on the controller fault handler, the controller might shut down.

- A watchdog time can range from 1 ms to 2,000,000 ms (2000 seconds). The default is 500 ms.
- The watchdog timer begins to time when the task is initiated and stops when all the programs within the task have executed.
- If the task takes longer than the watchdog time, a major fault occurs. (The time includes interruptions by other tasks.)
- A watchdog time-out fault (major fault) also occurs if a task is triggered again while it is executing (task overlap). This can happen if a lower-priority task is interrupted by a higher-priority task, delaying completion of the lower-priority task.
- You can use the controller fault handler to clear a watchdog fault. If the same watchdog fault occurs a second time during the same logic scan, the controller enters faulted mode, regardless of whether the controller fault handler clears the watchdog fault.

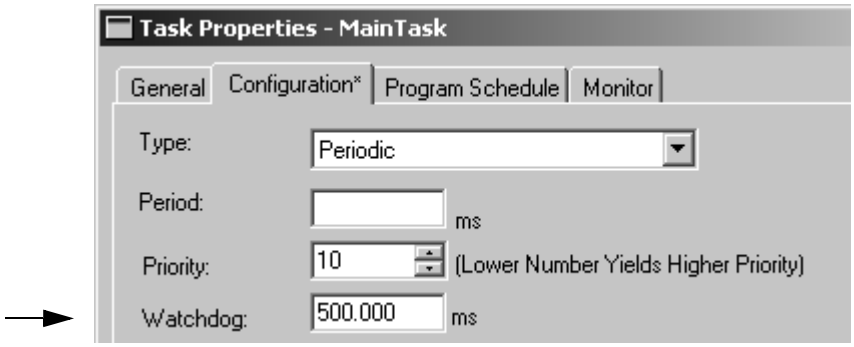
Adjust the Watchdog Timer for a Task

To change the watchdog time of a task, use the properties dialog box for the task.

1. In the controller organizer, right-click the task and select Properties.



2. Select the Configuration tab and select the watchdog timeout for the task, in milliseconds.



3. Click OK

Notes:

Designing a Sequential Function Chart

When to Use This Chapter

Use this chapter to design a **Sequential Function Chart (SFC)** for your process or system. An SFC is similar to a flowchart of your process. It defines the steps or states through which your system progresses. Use an SFC to:

- organize the functional specification for your system
- program and control your system as a series of steps and transitions

By using an SFC to specify your process, you gain these advantages:

- Since an SFC is a graphical representation of your process, it is easier to organize and read than a textual version. In addition, RSLogix 5000 software lets you:
  - add notes that clarify steps or capture important information for use later on
  - print the SFC to share the information with other individuals
- Since Logix5000 controllers support SFCs, you do not have to enter the specification a second time. You are programming your system as you specify it.

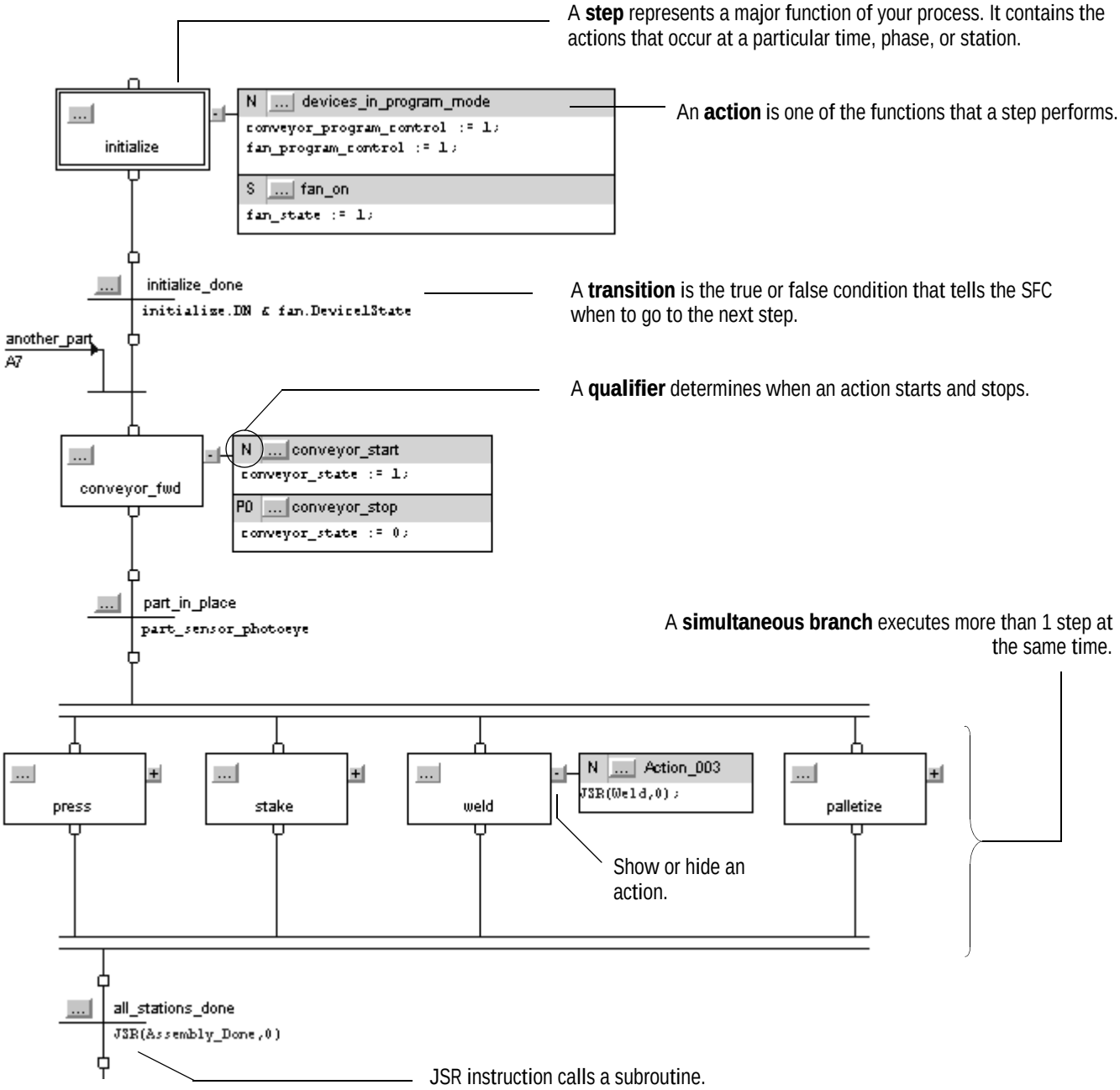
By using an SFC to program your process, you gain these advantages:

- graphical division of processes into its major logic pieces (steps)
- faster repeated execution of individual pieces of your logic
- simpler screen display
- reduced time to design and debug your program
- faster and easier troubleshooting
- direct access to the point in the logic where a machine faulted
- easy updates and enhancements

What is a Sequential Function Chart?

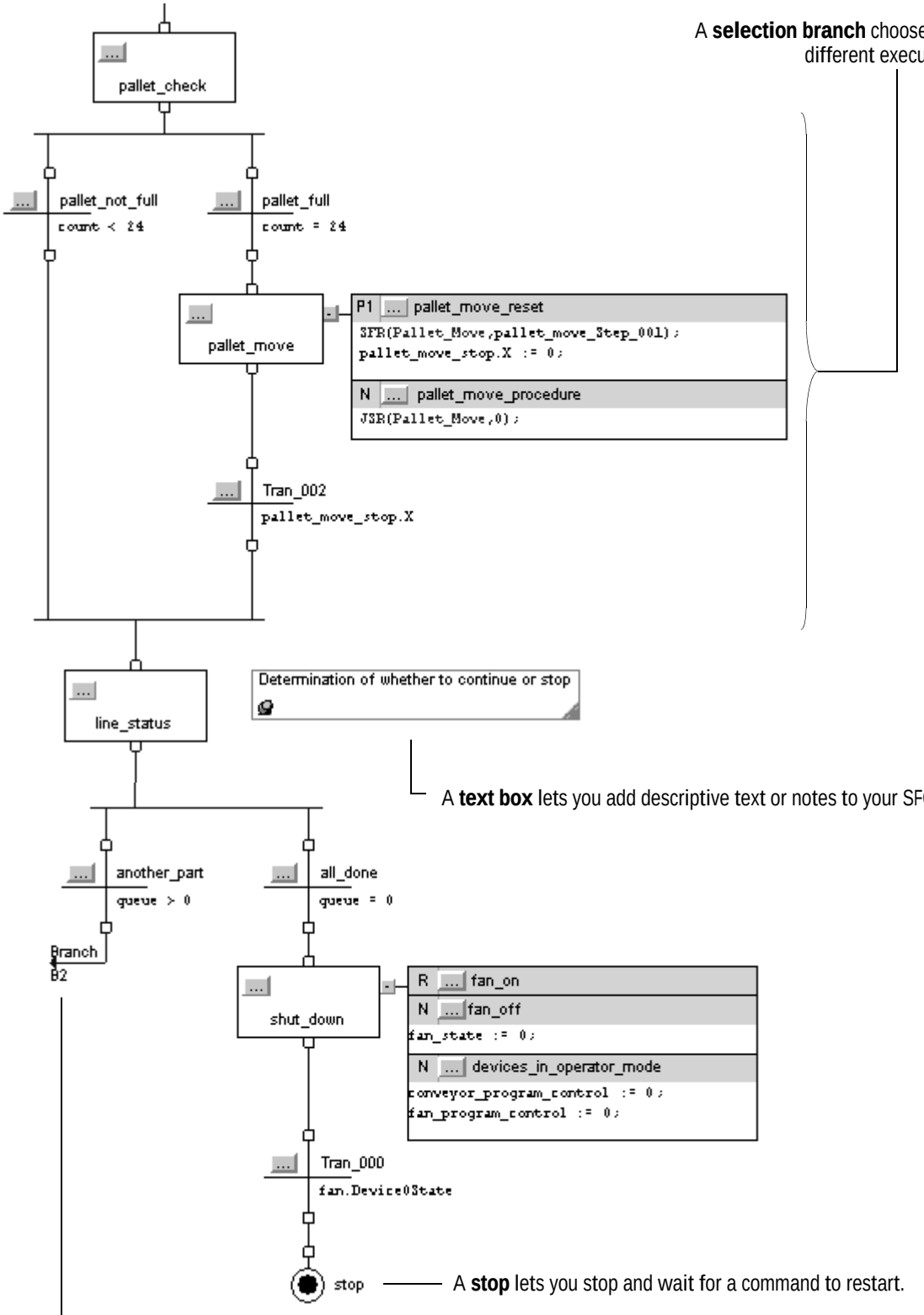
A S**Sequential Function Chart** (SFC) is similar to a flowchart. It uses steps and transitions to perform specific operations or actions. This example shows the elements of an SFC:

Figure 4.1 SFC Example



(continued on next page)

Figure 4.2 SFC Example (continued from previous page)



To design a sequential function chart:

Topic	Page
Define the Tasks	125
Choose How to Execute the SFC	126
Define the Steps of the Process	126
Organize the Steps	131
Add Actions for Each Step	135
Describe Each Action in Pseudocode	139
Choose a Qualifier for an Action	139
Define the Transition Conditions	140
Transition After a Specified Time	145
Turn Off a Device at the End of a Step	148
Keep Something On From Step-to-Step	154
End the SFC	158
Nest an SFC	161
Configure When to Return to the OS/JSR	162
Pause or Reset an SFC	163
Execution Diagrams	163

Define the Tasks

The first step in the development of an SFC is to separate the configuration and regulation of devices from the commands to those devices. Logix5000 controllers let you divide your project into one **continuous task** and multiple **periodic tasks** and **event tasks**.

1. Organize your project:

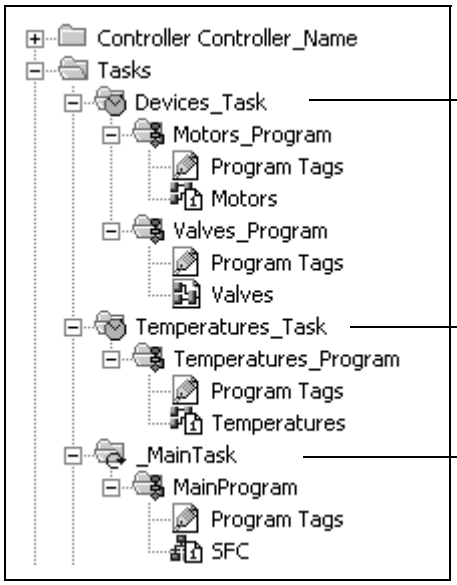
These functions:	Go here:
configure and regulate devices	periodic task
command a device to a specific state	SFC in the continuous task
sequence the execution of your process	

2. For those functions that go in a periodic task, group the functions according to similar update rates. Create a periodic task for each update rate.

   For example, 2-state devices may require faster updates than PID loops. Use separate periodic tasks for each.

In this example, a project uses two periodic tasks to regulate motors, valves, and temperature loops. An SFC to control the process.

EXAMPLE Define the Tasks



This task (periodic) uses Function Block diagrams to turn on or off motors and open or close valves. The SFC in MainTask commands the state for each device. The Function Block diagrams set and maintain that state.

This task (periodic) uses Function Block diagrams to configure and regulate temperature loops. The SFC in MainTask commands the temperatures. The Function Block diagrams set and maintain those temperatures.

This task (continuous) executes the sequential function chart (SFC). The SFC commands the specific state or temperature for each device or temperature loop.

Choose How to Execute the SFC

To execute an SFC, either configure it as the main routine for a program or call it as a subroutine.

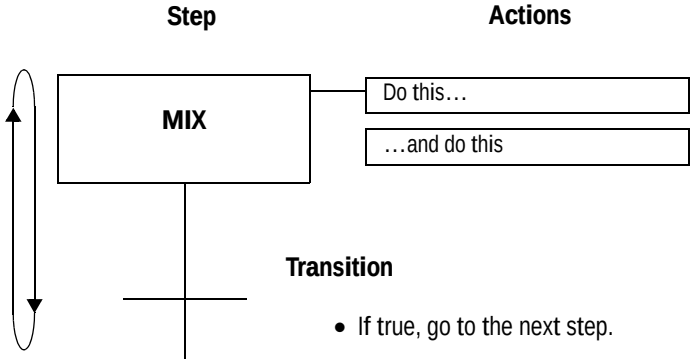
If:	Then:
The SFC is the only routine in the program.	Configure the SFC as the main routine for the program.
The SFC calls *all* the other routines of the program.	
The program requires other routines to execute independent of the SFC.	1. Configure another routine as the main routine for the program. 2. Use the main routine to call the SFC as a subroutine.
The SFC uses boolean actions.	

If the SFC uses boolean actions, then other logic must run independent of the SFC and monitor status bits of the SFC.

Define the Steps of the Process

A **step** represents a major function of your process. It contains the actions that occur at a particular time, phase, or station.

A step executes continuously until a transition tells the SFC to go to the next step.

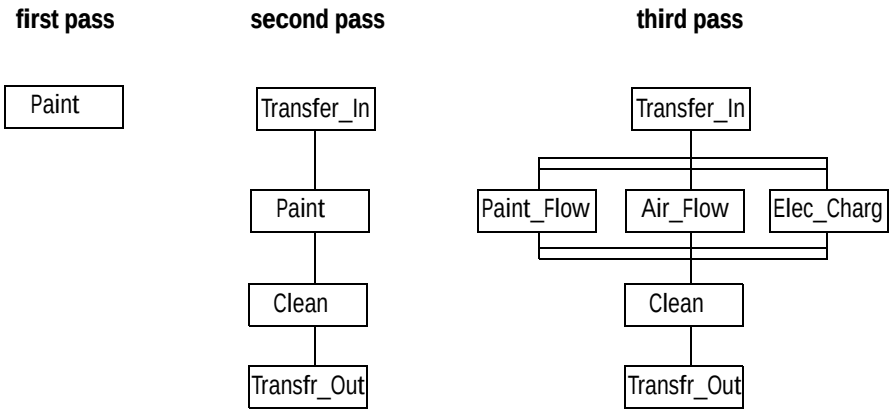


A **transition** ends a step. The transition defines the physical conditions that must occur or change in order to go to the next step.

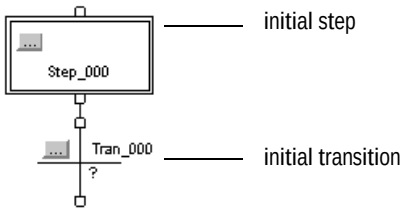
Step Guidelines

As you define the steps of your process, follow these guidelines:

- Start with large steps and refine the steps in several passes.



- When you first open an SFC routine, it contains an initial step and transition. Use this step to initialize your process.



  The controller executes the initial step:

  – after a project download when the controller goes into Run mode.
  – if the SFC is configured to, when the controller transitions to Run mode and on power-up
  – when the routine containing the chart is modified online and the controller transitions to or from Test mode

- To identify a step, look for a physical change in your system, such as new part that is in position, a temperature that is reached, a preset time that is reached, or a recipe selection that occurs. The step is the actions that take place before that change.

- Stop when your steps are in meaningful increments. For example:

This organization of steps:	Is:
produce_solution	probably too large
set_mode, close_outlet, set_temperature, open_inlet_a, close_inlet_a, set_timer, reset_temperature, open_outlet, reset_mode	probably too small
preset_tank, add_ingredient_a, cook, drain	probably about right

SFC_STEP Structure

Each step uses a tag to provide information about the step. Access this information via either the Step Properties dialog box or the Monitor Tags tab of the Tags window:

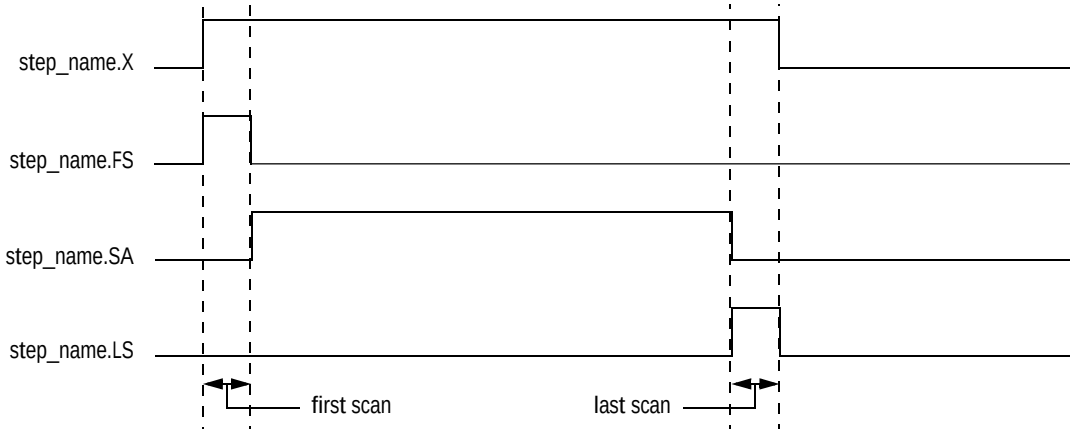
If you want to:	Then check or set this member:	Data type:	Details:
determine how long a step has been active (milliseconds)	T	DINT	When a step becomes active, the Timer (T) value resets and then starts to count up in milliseconds. The timer continues to count up until the step goes inactive, regardless of the Preset (PRE) value.
flag when the step has been active for a specific length of time (milliseconds)	PRE	DINT	Enter the time in the Preset (PRE) member. When the Timer (T) reaches the Preset value, the Done (DN) bit turns on and stays on until the step becomes active again. As an option, enter a numeric expression that calculates the time at runtime.
	DN	BOOL	When the Timer (T) reaches the Preset (PRE) value, the Done (DN) bit turns on and stays on until the step becomes active again.
flag if a step did not execute long enough	LimitLow	DINT	Enter the time in the LimitLow member (milliseconds). • If the step goes inactive before the Timer (T) reaches the LimitLow value, the AlarmLow bit turns on. • The AlarmLow bit stays on until you reset it. • To use this alarm function, turn on (check) the AlarmEnable (AlarmEn) bit. As an option, enter a numeric expression that calculates the time at runtime.
	AlarmEn	BOOL	To use the alarm bits, turn on (check) the AlarmEnable (AlarmEn) bit.
	AlarmLow	BOOL	If the step goes inactive before the Timer (T) reaches the LimitLow value, the AlarmLow bit turns on. • The bit stays on until you reset it. • To use this alarm function, turn on (check) the AlarmEnable (AlarmEn) bit.

If you want to:	Then check or set this member:	Data type:	Details:
flag if a step is executing too long	LimitHigh	DINT	Enter the time in the LimitHigh member (milliseconds). • If the Timer (T) reaches the LimitHigh value, the AlarmHigh bit turns on. • The AlarmHigh bit stays on until you reset it. • To use this alarm function, turn on (check) the AlarmEnable (AlarmEn) bit. As an option, enter a numeric expression that calculates the time at runtime.
	AlarmEn	BOOL	To use the alarm bits, turn on (check) the AlarmEnable (AlarmEn) bit.
	AlarmHigh	BOOL	If the Timer (T) reaches the LimitHigh value, the AlarmHigh bit turns on. • The bit stays on until you reset it. • To use this alarm function, turn on (check) the AlarmEnable (AlarmEn) bit.
do something while the step is active (including first and last scan)	X	BOOL	The X bit is on the entire time the step is active (executing). Typically, we recommend that you use an action with a N Non-Stored qualifier to accomplish this.
do something one time when the step becomes active	FS(1)	BOOL	The FS bit is on during the first scan of the step. Typically, we recommend that you use an action with a P1 Pulse (Rising Edge) qualifier to accomplish this.
do something while the step is active, *except* on the first and last scan	SA	BOOL	The SA bit is on when the step is active except during the first and last scan of the step.
do something one time on the last scan of the step	LS(1)	BOOL	The LS bit is on during the last scan of the step. Use this bit only if you do the following: On the Controller Properties dialog box, SFC Execution tab, set the Last Scan of Active Step to Don't Scan or Programmatic reset. Typically, we recommend that you use an action with a P0 Pulse (Falling Edge) qualifier to accomplish this.
determine the target of an SFC Reset (SFR) instruction	Reset	BOOL	An SFC Reset (SFR) instruction resets the SFC to a step or stop that the instruction specifies. • The Reset bit indicates to which step or stop the SFC will go to begin executing again. • Once the SFC executes, the Reset bit clears.
determine the maximum time that a step has been active during any of its executions	TMax	DINT	Use this for diagnostic purposes. The controller clears this value only when you select the Restart Position of Restart at initial step and the controller changes modes or experiences a power cycle.
determine if the Timer (T) value rolls over to a negative value	OV	BOOL	Use this for diagnostic purposes.

If you want to:	Then check or set this member:	Data type:	Details:
determine how many times a step has become active	Count	DINT	This is not a count of scans of the step. • The count increments each time the step becomes active. • It increments again only after the step goes inactive and then active again. • The count resets only if you configure the SFC to restart at the initial step. With that configuration, it resets when the controller changes from program mode to run mode.
use one tag for the various status bits of this step	Status	DINT	**For this member: / Use this bit:** Reset 22 AlarmHigh 23 AlarmLow 24 AlarmEn 25 OV 26 DN 27 LS 28 SA 29 FS 30 X 31

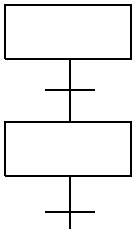
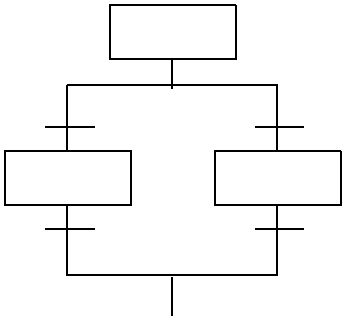
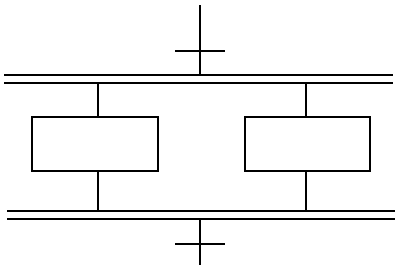
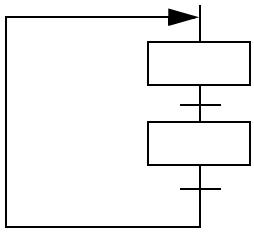
(1) The FS and LS bits are only active during a step's execution. Once a step finishes executing the code within its actions, the FS and/or LS bits are reset. If you reference either of these bits in code outside of the SFC routine in a different part of the project, the bits are always cleared (0).

This diagram shows the relationship of the X, FS, SA, and LS bits.

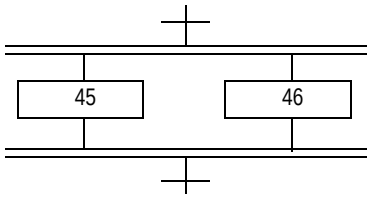
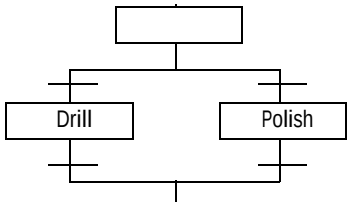
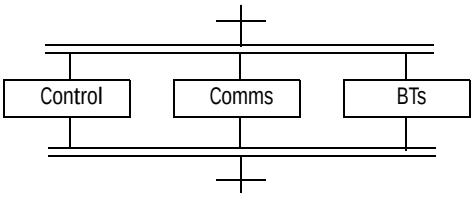
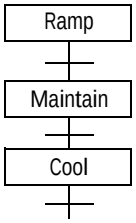
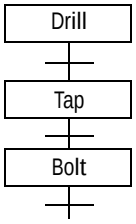
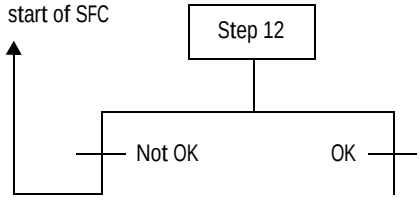


Organize the Steps

Once you define the steps of your process, organize them into sequences, simultaneous branches, selection branches, or loops.

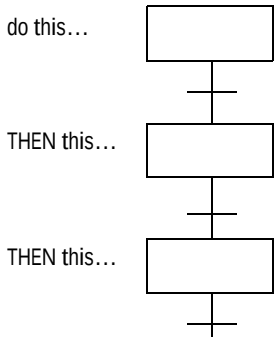
To:	Use this structure:	With these considerations:
Execute 1 or more steps in sequence: • One executes repeatedly. • Then the next executes repeatedly.	**Sequence** 	The SFC checks the transition at the end of the step: • If true, the SFC goes to the next step. • If false, the SFC repeats the step.
• Choose between alternative steps or groups of steps depending on logic conditions • Execute a step or steps or skip the step or steps depending on logic conditions	**Selection Branch** 	• It is OK for a path to have no steps and only a transition. This lets the SFC skip the selection branch. • By default, the SFC checks from left to right the transitions that start each path. It takes the first true path. • If no transitions are true, the SFC repeats the previous step. • RSLogix 5000 software lets you change the order in which the SFC checks the transitions.
Execute 2 or more steps at the same time. All paths must finish before continuing the SFC	**Simultaneous Branch** 	• A single transition ends the branch. • The SFC checks the ending transition after the last step in each path has executed at least once. If the transition is false, the SFC repeats the previous step.
Loop back to a previous step	**Wire to a Previous Step** 	• Connect the wire to the step or simultaneous branch to which you want to go. • Do not wire into, out of, or between a simultaneous branch.

Here are some examples of SFC structures for different situations:

Example situation:	Example solution:
Station 45 and 46 of an assembly line work on parts simultaneously. When both stations are done, the parts move down 1 station.	**Simultaneous Branch** 
Depending on the build code, a station either drills or polishes.	**Selection Branch** 
To simplify my programming, I want to separate communications and block transfers from other control logic. All occur at the same time.	**Simultaneous Branch** 
In a heat treating area, the temperature ramps up at a specific rate, maintains that temperature for a specific duration, and then cools at a specific rate.	**Sequence** 
At station 12, the machine drills, taps, and bolts a part. The steps occur one after the other.	**Sequence** 
Step 12 inspects a process for the correct mix of chemicals. • If OK, then continue with the remaining steps. • If not OK, go to the top of the SFC and purge the system.	**Wire** 

Sequence

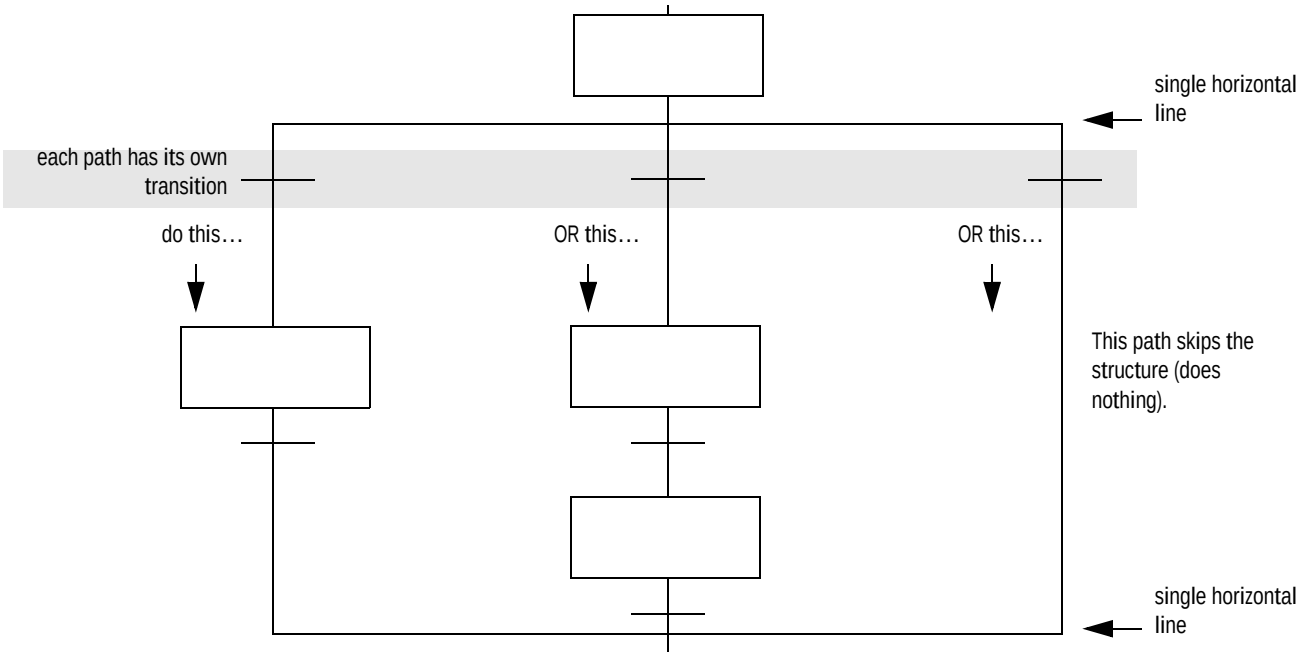
A sequence is a group of steps that execute one after the other.



Selection Branch

A selection branch represents a choice between one path (step or group of steps) or another path (i.e., an OR structure).

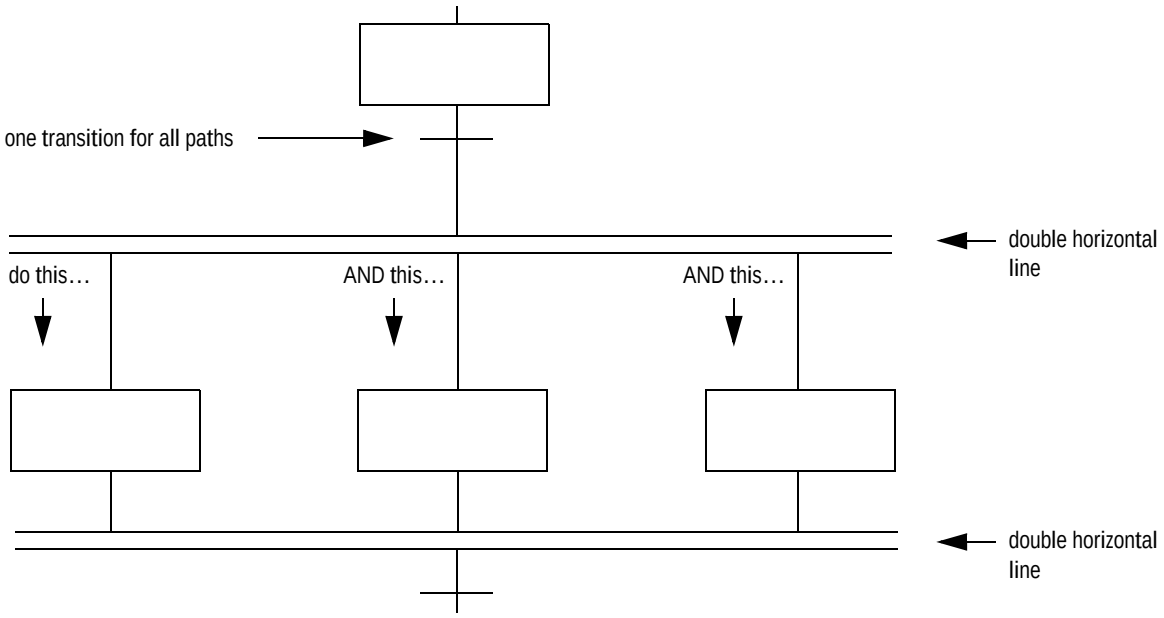
- Only one path executes.
- By default the SFC checks the transitions from left to right.
  - The SFC takes the first true path.
  - RSLogix 5000 software lets you change the order in which the SFC checks the transitions (see chapter 5).



Simultaneous Branch

A simultaneous branch represents paths (steps or group of steps) that occur at the same time (i.e., an AND structure).

- All paths execute.
- All paths must finish before continuing with the SFC.
- The SFC checks the transition after the last step of each path has executed at least once.

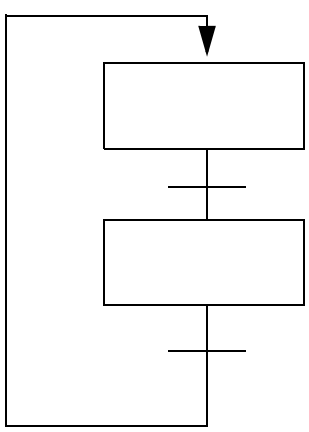


Wire to a Previous Step

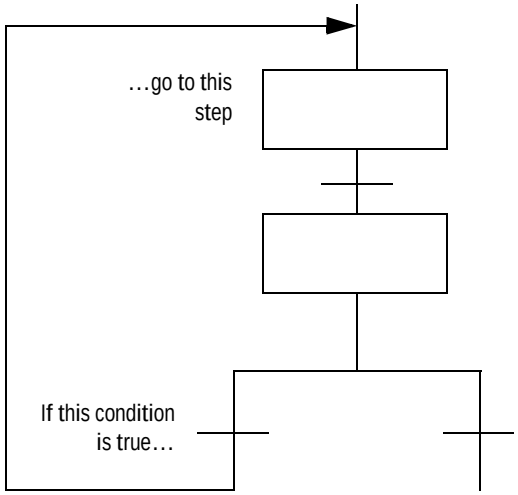
In addition to connecting steps in sequences, simultaneous branches, and selection branches, you can connect a step to a previous point in your SFC. This lets you:

- loop back and repeat steps
- return to the beginning of the SFC and start over

For example:



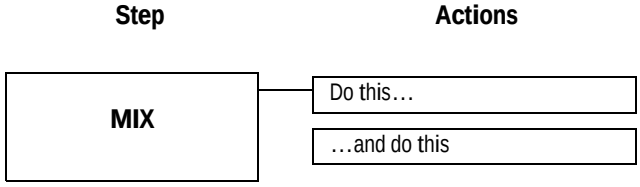
simple loop that repeats the entire SFC



path of a selection branch that returns to a previous step

Add Actions for Each Step

Use **actions** to divide a step into the different functions that the step performs, such as commanding a motor, setting the state of a valve, or placing a group of devices in a specific mode.



How Do You Want to Use the Action?

There are two types of actions:

If you want to:	Then:
execute Structured Text directly in the SFC	Use a Non-Boolean Action
call a subroutine	
use the automatic reset option to reset data upon leaving a step	
only set a bit and program other logic to monitor the bit to determine when to execute.	Use a Boolean Action

Use a Non-Boolean Action

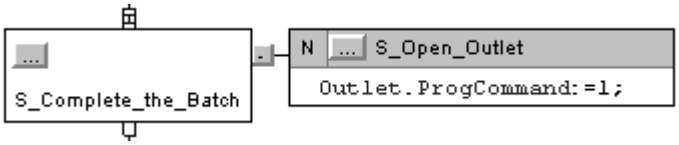
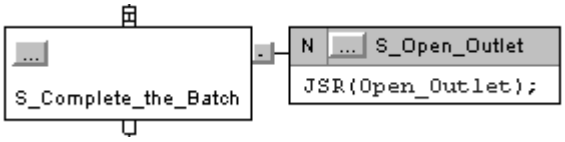
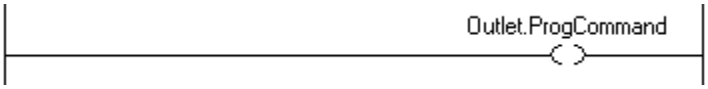
A non-boolean action contains the logic for the action. It uses Structured Text to execute assignments and instructions or call a subroutine.

With non-boolean actions, you also have the option to **postscan** (automatically reset) the assignments and instructions before leaving a step:

- During postscan the controller executes the assignments and instructions as if all conditions are false.
- The controller postscans both embedded Structured Text and any subroutine that the action calls.

To automatically reset assignments and instructions, see "Turn Off a Device at the End of a Step" on page 4-148.

To program a non-boolean action, you have these options:

If you want to:	Then:
• execute your logic without additional routines • use Structured Text assignments, constructs, and instructions	Embed Structured Text. For example:  When the S_Complete_the_Batch step is active, the S_Open_Outlet action executes. The action sets the Outlet.ProgCommand tag equal to 1, which opens the outlet valve.
• re-use logic in multiple steps • use another language to program the action, such as ladder logic • nest an SFC	Call a subroutine. For example:  When the S_Complete_the_Batch step is active, the S_Open_Outlet action executes. The action calls the Open_Outlet routine. **Open_Outlet Routine**  When the Open_Outlet routine executes, the OTE instruction sets the Outlet.ProgCommand tag equal to 1, which opens the outlet valve.

You cannot reuse a non-boolean action within the same SFC except to reset a stored action. Only one instance of a specific non-boolean action is permitted per SFC.

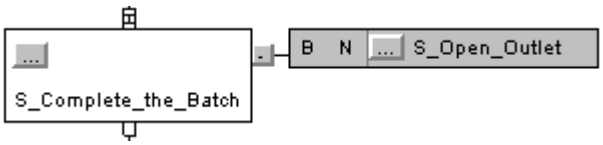
Use a Boolean Action

A boolean action contains no logic for the action. It simply sets a bit in its tag (SFC_ACTION structure). To do the action, other logic must monitor the bit and execute when the bit is on.

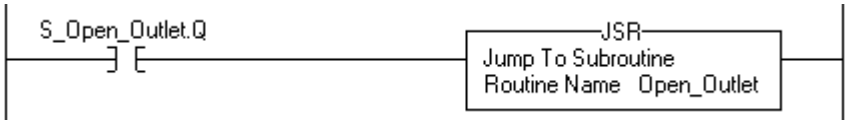
With boolean actions, you have to manually reset the assignments and instructions that are associated with the action. Since there is no link between the action and the logic that performs the action, the automatic reset option does not effect boolean actions.

For example:

EXAMPLE



When the S_Complete_the_Batch step is active, the S_Open_Outlet action executes. When the action is active, its Q bit turns on.



A ladder Logic routine monitors the Q bit (S_Open_Outlet.Q). When the Q bit is on, the JSR instruction executes and opens the outlet valve.

You can reuse a boolean action multiple times within the same SFC.

SFC_ACTION Structure

Each action (non-boolean and boolean) uses a tag to provide information about the action. Access this information via either the Action Properties dialog or the Monitor Tags tab of the Tags window:

If you want to:	Then check or set this member:	Data type:	Details:	
determine when the action is active	Q	BOOL	The status of the Q bit depends on whether the action is a boolean action or non-boolean action:	
			If the action is:	**Then the Q bit is:**
			boolean	on (1) the entire time the action is active, including the last scan of the action
			non-boolean	on (1) while the action is active but off (0) at the last scan of the action
			To use a bit to determine when an action is active, use the Q bit.	
	A	BOOL	The A bit is on the entire time the action is active.	
determine how long an action has been active (milliseconds)	T	DINT	When an action becomes active, the Timer (T) value resets and then starts to count up in milliseconds. The timer continues to count up until the action goes inactive, regardless of the Preset (PRE) value.	
use one of these time-based qualifiers: L, SL, D, DS, SD	PRE	DINT	Enter the time limit or delay in the Preset (PRE) member. The action starts or stops when the Timer (T) reaches the Preset value. As an option, enter a numeric expression that calculates the time at runtime.	
determine how many times an action has become active	Count	DINT	This is *not* a count of scans of the action. • The count increments each time the action becomes active. • It increments again only after the action goes inactive and then active again. • The count resets only if you configure the SFC to restart at the initial step. With that configuration, it resets when the controller changes from program mode to run mode.	
use one tag for the various status bits of this action	Status	DINT	**For this member:**	**Use this bit:**
			Q	30
			A	31

Describe Each Action in Pseudocode

To organize the logic for an action, first describe the action in pseudocode. If you are unfamiliar with pseudocode:

- Use a series of short statements that describe what should happen.
- Use terms or symbols such as: if, then, otherwise, until, and, or, =, >, <.
- Sequence the statements in the order that they should execute.
- If necessary, name the conditions to check first (when 1st) and then the action to take second (what 2nd).

Enter the pseudocode into the body of the action. After you enter the pseudocode, you can:

- Refine the pseudocode so it executes as Structured Text.
- Use the pseudocode to design your logic and leave the pseudocode as comments. Since all Structured Text comments download to the controller, your pseudocode is always available as documentation for the action.

To convert the pseudocode to Structured Text comments, add these comment symbols:

For a comment:	Use one of these formats:
on a single line	`//comment`
that spans more than one line	`(*start of comment . . . end of comment*)` `/*start of comment . . . end of comment*/`

Choose a Qualifier for an Action

Each action (non-boolean and boolean) uses a **qualifier** to determine when it starts and stops.

The default qualifier is Non-Stored. The action starts when the step is activated and stops when the step is deactivated.

To change when an action starts or stops, assign a different qualifier:

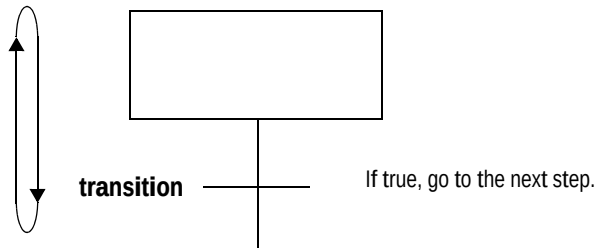
Table 4.1 Choose a Qualifier for an Action

If you want the action to:	And:	Then assign this qualifier:	Which means:
start when the step is activated	stop when the step is deactivated	N	Non-Stored
	execute only once	P1	Pulse (Rising Edge)
	stop before the step is deactivated or when the step is deactivated	L	Time Limited
	stay active until a Reset action turns off this action	S	Stored
	stay active until a Reset action turns off this action or a specific time expires, even if the step is deactivated	SL	Stored and Time Limited
start a specific time after the step is activated *and* the step is still active	stop when the step is deactivated	D	Time Delayed
	stay active until a Reset action turns off this action	DS	Delayed and Stored
start a specific time after the step is activated, even if the step is deactivated before this time	stay active until a Reset action turns off this action	SD	Stored and Time Delayed
execute once when the step is activated	execute once when the step is deactivated	P	Pulse
start when the step is deactivated	execute only once	P0	Pulse (Falling Edge)
turn off (reset) a stored action: • S Stored • SL Stored and Time Limited • DS Delayed and Stored • SD Stored and Time Delayed	——————————→	R	Reset

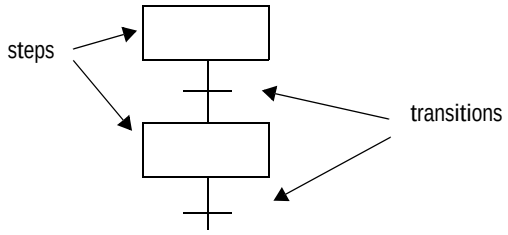
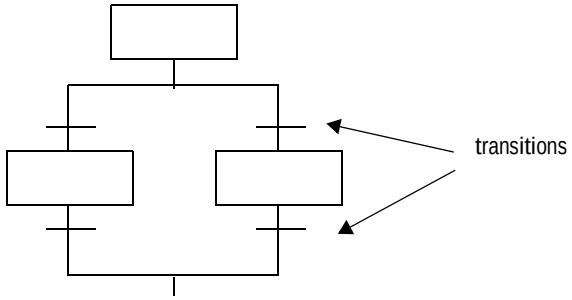
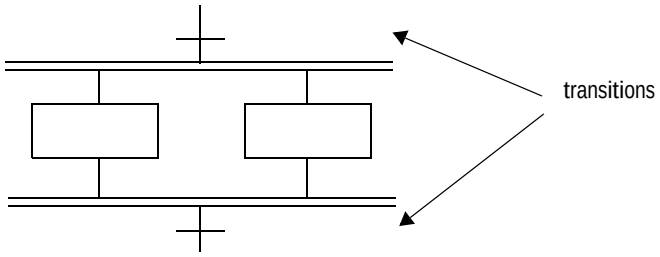
Define the Transition Conditions

The transition is the physical conditions that must occur or change in order to go to the next step.

The transition tells the SFC when to go to the next step.



Transitions occur in the:

For this structure:	Make sure that:
sequence	A transition is between each step. 
selection branch	Transitions are inside the horizontal lines. 
simultaneous branch	Transitions are outside the horizontal lines. 

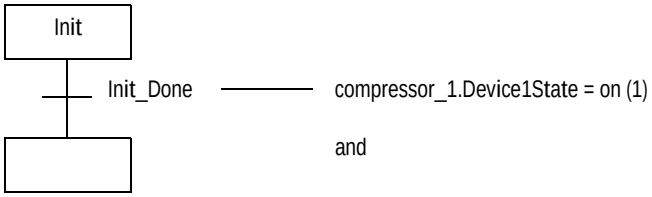
Here are two examples of transitions:

EXAMPLE

You want to:

a. Turn on 2 compressors. When a compressor is on, the Device1State bit is on.
b. When both compressors are on, go to the next step.

Solution:

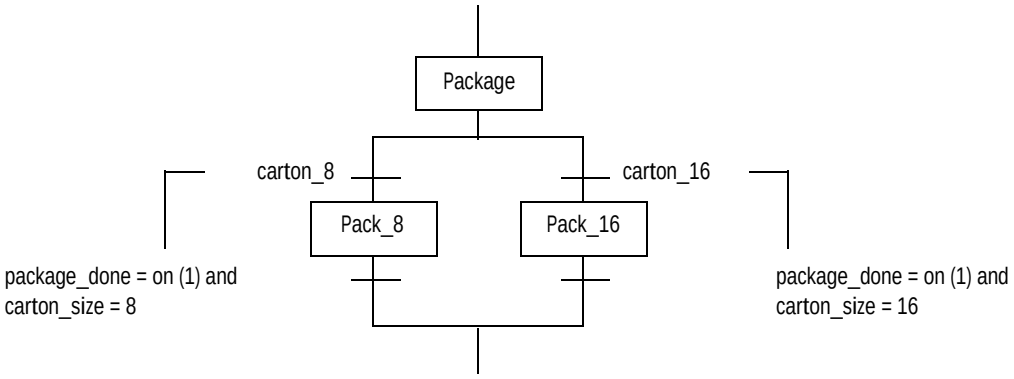


EXAMPLE

You want to:

a. Package the product. When the product is in the package, the package_done bit turns on.
b. Pack the product either 8 per carton or 16 per carton.

Solution:



To override the state of a transition, see "Forcing Logic Elements" on page 15-387.

Transition Tag

Each transition uses a BOOL tag to represent the true or false state of the transition.

If the transition is:	The value is:	And:
true	1	The SFC goes to the next step.
false	0	The SFC continues to execute the current step.

How Do You Want to Program the Transition?

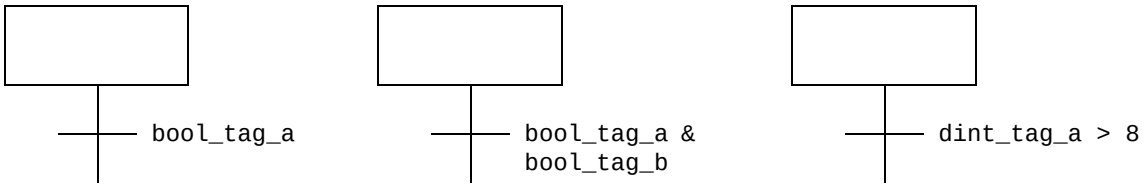
To program the transition, you have these options:

If you want to:	Then:
enter the conditions as an expression in Structured Text	Use a BOOL Expression
enter the conditions as instructions in another routine	Call a Subroutine
use the same logic for multiple transitions	

Use a BOOL Expression

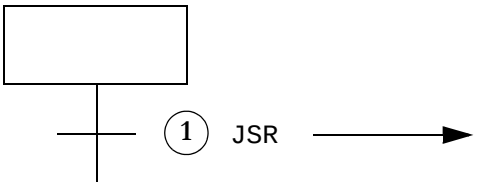
The simplest way to program the transition is to enter the conditions as a **BOOL expression** in Structured Text. A BOOL expression uses bool tags, relational operators, and logical operators to compare values or check if conditions are true or false. For example, `tag1>65`.

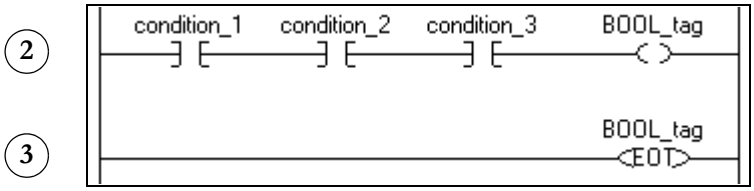
Here are some examples of BOOL expressions.



Call a Subroutine

To use a subroutine to control a transition, include an End Of Transition (EOT) instruction in the subroutine. The EOT instruction returns the state of the conditions to the transition, as shown below.





- or -

2

```
If condition_1 & condition_2 &
condition_3 then
     BOOL_tag := 1;
Else
     BOOL_tag := 0;
End_if;
EOT(BOOL_tag);
```

3

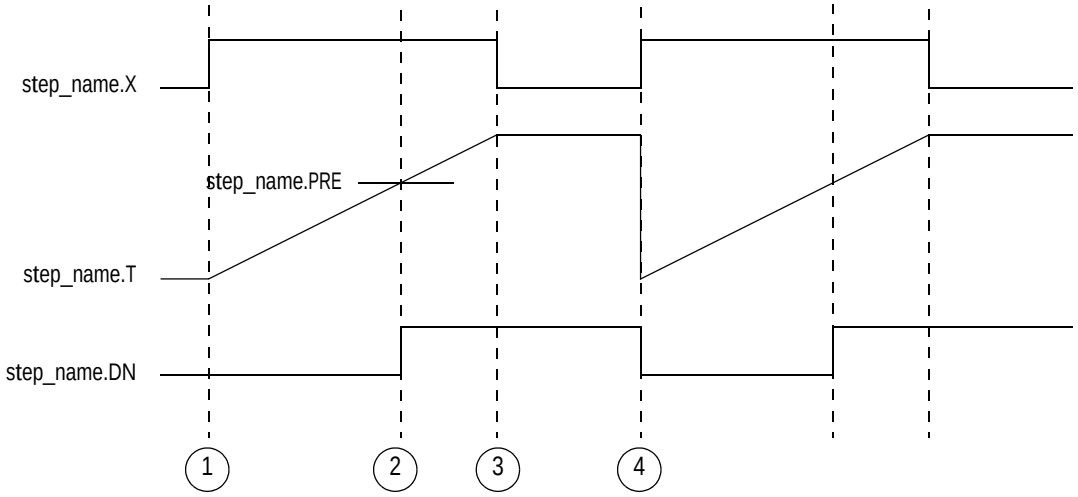
1. Call a subroutine.
2. Check for the required conditions. When those conditions are true, turn on a BOOL tag.
3. Use an EOT instruction to set the state of the transition equal to the value of the BOOL tag. When the BOOL tag is on (true), the transition is true.

Transition After a Specified Time

Each step of the SFC includes a millisecond timer that runs whenever the step is active. Use the timer to:

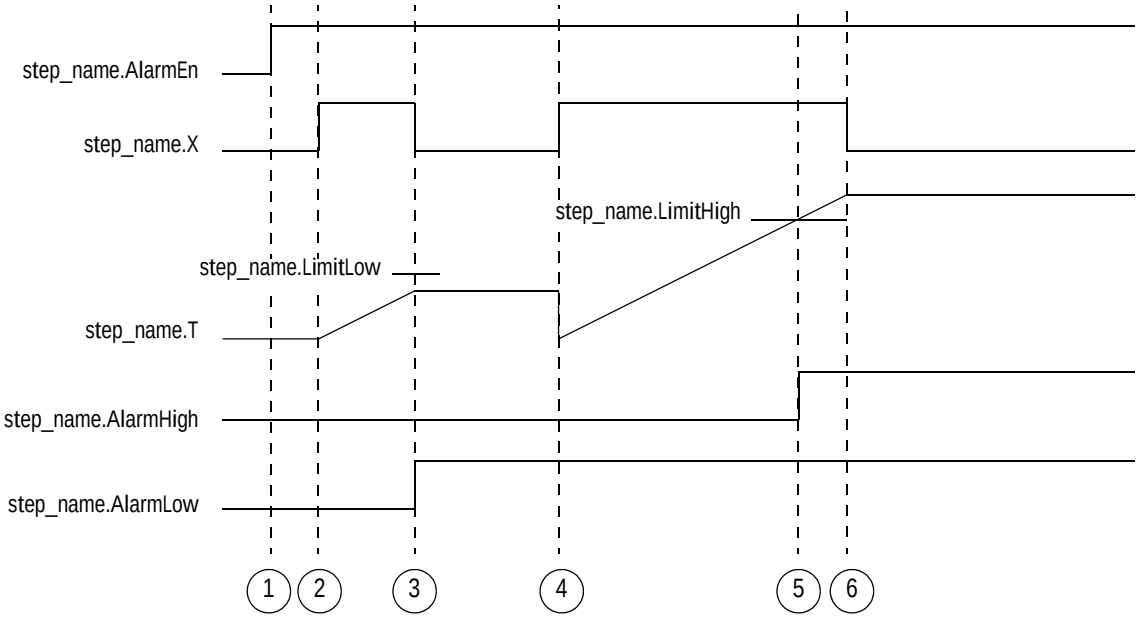
- signal when the step has run for the required time and the SFC should go to the next step
- signal when the step has run too long and the SFC should go to an error step

Figure 4.3 Action of a timer and associated bits of a step:



Description:
1. Step becomes active. X bit turns on. Timer (T) begins to increment.
2. Timer reaches the Preset (PRE) value of the step. DN bit turns on. Timer continues to increment.
3. Step becomes inactive. X bit turns off. Timer retains its value. DN remains on.
4. Step becomes active. X bit turns on. Timer clears and then begins to increment. DN bit turns off.

Figure 4.4 Action of the low and high alarms for a step:



Description:
1. AlarmEn is on. To use the low and high alarms turn this bit on. Turn the bit on via the properties dialog box or the tag for the step.
2. Step becomes active. X bit turns on. Timer (T) begins to increment.
3. Step becomes inactive. X bit turns off. Timer retains its value. Since Timer is less than LimitLow, AlarmLow bit turns on.
4. Step becomes active. X bit turns on. Timer clears and then begins to increment. AlarmLow stays on. (You have to manually turn it off.)
5. Timer reaches the LimitHigh value of the step. AlarmHigh bit turns on. Timer continues to increment.
6. Step becomes inactive. X bit turns off. Timer retains its value. AlarmHigh stays on. (You have to manually turn it off.)

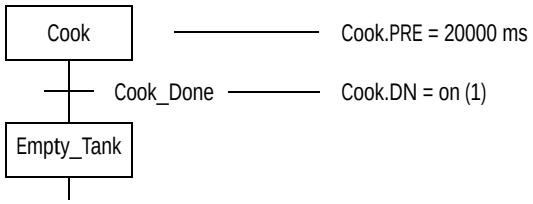
Here is an example of the use of the Preset time of a step.

EXAMPLE

Functional specification says:

a. Cook the ingredients in the tank for 20 seconds.
b. Empty the tank.

Solution:



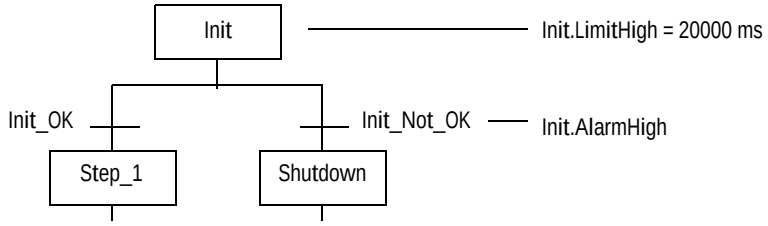
Here is an example of the use of the high alarm of a step.

EXAMPLE

Functional specification says:

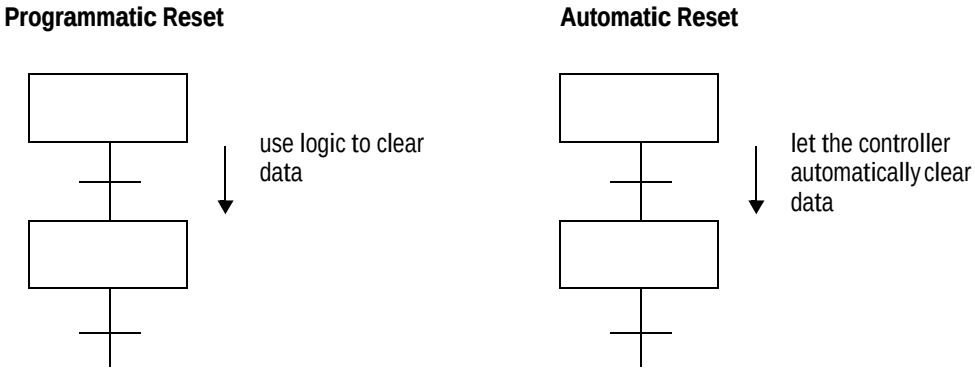
a. Home 8 devices.
b. If all 8 devices are not home within 20 seconds, then shutdown the system.

Solution:



Turn Off a Device at the End of a Step

When the SFC leaves a step, you have several options on how to turn off devices that the step turned on.



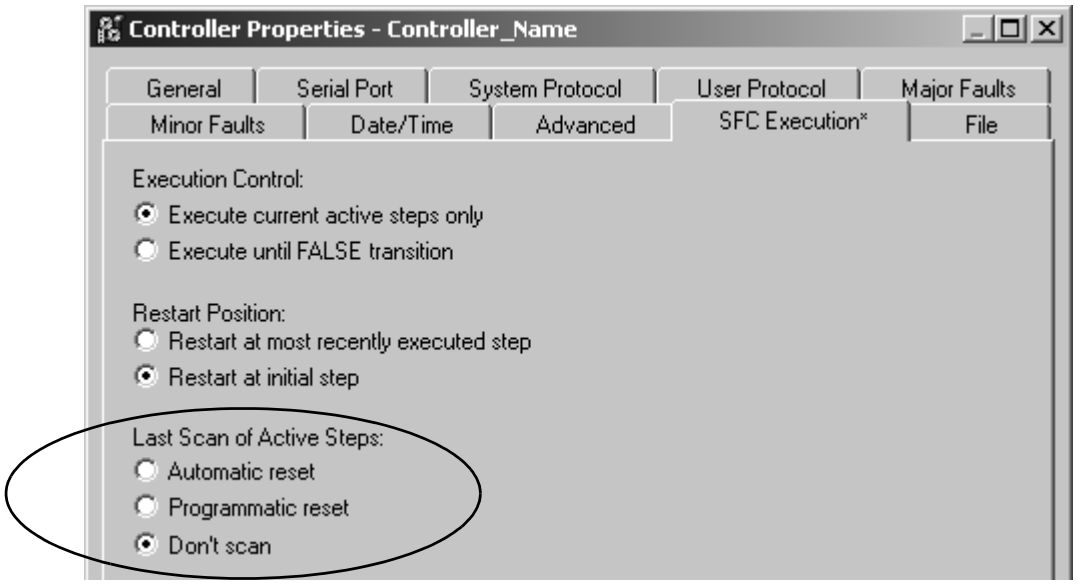
Each option requires you to make these choices:

1. Choose a last scan option.
2. Based on the last scan option, develop your logic so that the last scan returns data to the desired values.

Choose a Last Scan Option

On the last scan of each step, you have these options. The option that you choose applies to all steps in all SFCs of this controller.

If you want to:	And on the last scan of a step:	Then:	See:
control which data to clear	Execute only P and P0 actions and use them to clear the required data.	Use the Don't Scan Option	page 4-150
	Execute all actions and use either of these options to clear the required data: • status bits of the step or action to condition logic • P and P0 actions	Use the Programmatic Reset Option	page 4-151
let the controller clear data	→	Use the Automatic Reset Option	page 4-153



This table compares the different options for handling the last scan of a step:

Characteristic:	**During the last scan of a step, this option does:**		
	Don't scan	**Programmatic reset**	**Automatic reset**
execution actions	Only P and P0 actions execute. They execute according to their logic.	All actions execute according to their logic.	• P and P0 actions execute according to their logic. • All other actions execute in Postscan mode. • On the next scan of the routine, the P and P0 actions execute in Postscan mode.
retention of data values	All data keeps its current values.	All data keeps its current values.	• Data reverts to its values for postscan. • Tags to the left of [:=] assignments clear to zero.
method for clearing data	Use P and P0 actions.	Use either: • status bits of the step or action to condition logic • P and P0 actions	Use either: • [:=] assignment (non-retentive assignment) • instructions that clear their data during postscan
reset of a nested SFC	A nested SFCs remains at its current step.	A nested SFCs remains at its current step.	For the Restart Position property, if you choose the Restart at initial step option, then: • A nested SFC resets to its initial step. • The X bit of a stop element in a nested SFC clears to zero.

Use the Don't Scan Option

The default option for handling the last scan of a step is *Don't scan*. With this option, all data keeps its current values when the SFC leaves a step. This requires you to use additional assignments or instructions to clear any data that you want to turn off at the end of a step.

To turn off a device at the end of a step:

1. Make sure that the Last Scan of Active Steps property is set to the Don't scan option (default).

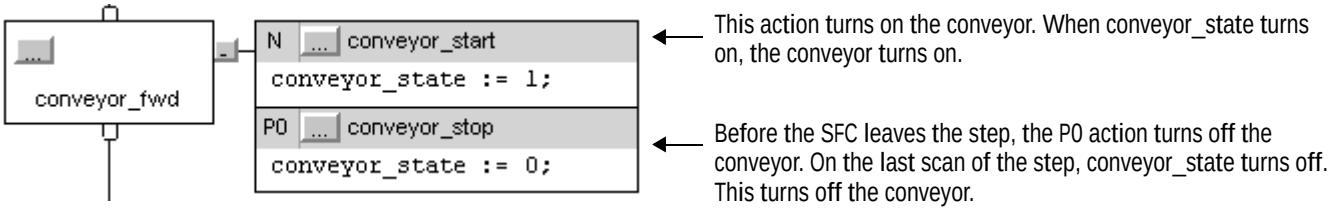
2. Use a P0 Pulse (Falling Edge) action to clear the required data. Make sure that the P0 action or actions are last in the order of actions for the step.

During the last scan of the step, the Don't scan option executes only P and P0 actions. The assignments and instructions of the actions execute according to their logic conditions.

- The controller *does not* execute a **postscan** of assignments or instructions.
- When the SFC leaves the step, all data keeps its current values.

This example uses an action to turn on a conveyor at the start of a step. A different action turns off the conveyor at the end of the step.

EXAMPLE Use the Don't Scan Option



Use the Programmatic Reset Option

An optional method to programmatically turn off (clear) devices at the end of a step is to execute all actions on the last scan of the step. This lets you execute your normal logic as well as turn off (clear) devices at the end of a step.

1. In the Last Scan of Active Steps property, select the Programmatic reset option:

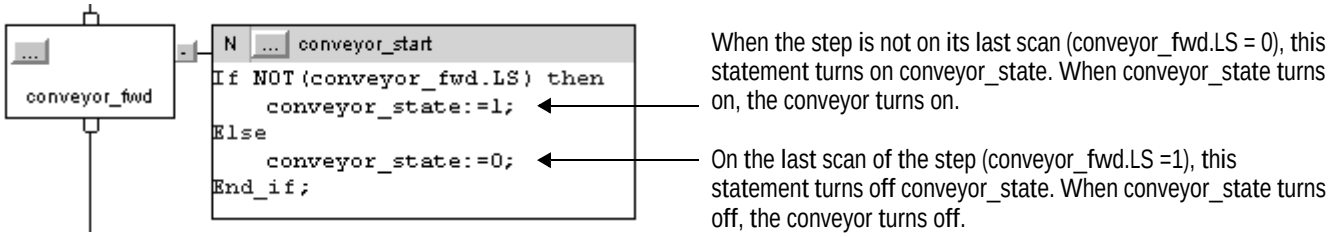
2. Clear the required data using any of these methods:
   - To your normal logic, add logic that clears the required data. Use the LS bit of the step or the Q bit of the action to condition the execution of the logic.
   - Use a P0 Pulse (Falling Edge) action to clear the required data. Make sure that the P0 action or actions are last in the order of actions for the step.

During the last scan of the step, the Programmatic reset option executes all assignments and instructions according to logic conditions.

- The controller does not **postscan** the assignments or instructions.
- When the SFC leaves the step, all data keeps its current value.

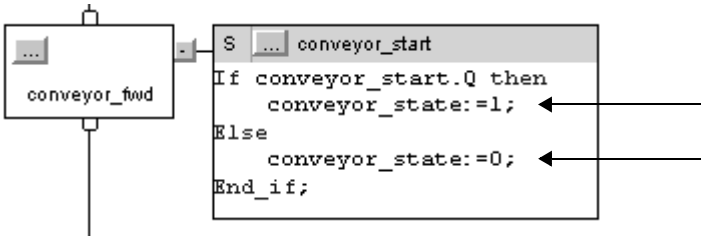
This example uses a single action to turn on and off a conveyor. The LS bit of the step conditions the execution of the logic. See "SFC_STEP Structure" on page 4-128.

EXAMPLE Use the Programmatic Reset Option and the LS Bit



For an action that uses one of the stored qualifiers, use the Q bit of the action to condition your logic. See "SFC_ACTION Structure" on page 4-138.

EXAMPLE Use the Programmatic Reset Option and the Q Bit

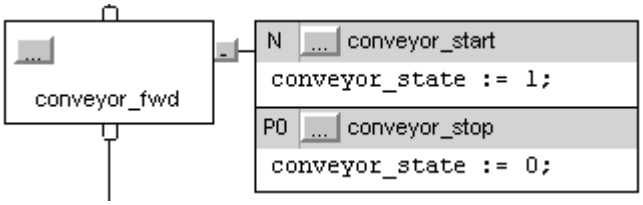


When the action is not on its last scan (conveyor_start.Q =1), this statement turns on conveyor_state. When conveyor_state turns on, the conveyor turns on.

On the last scan of the action (conveyor_start.Q =0), this statement turns off conveyor_state. When conveyor_state turns off, the conveyor turns off.

You can also use a P0 Pulse (Falling Edge) action to clear data. This example uses an action to turn on a conveyor at the start of a step. A different action turns off the conveyor at the end of the step.

EXAMPLE Use the Programmatic Reset Option and a P0 Action



This action turns on the conveyor. When conveyor_state turns on, the conveyor turns on.

Before the SFC leaves the step, the P0 action turns off the conveyor. On the last scan of the step, conveyor_state turns off. This turns off the conveyor.

Use the Automatic Reset Option

To automatically turn off (clear) devices at the end of a step:

1. In the Last Scan of Active Steps property, select the Automatic reset option.

2. To turn off a device at the end of the step, control the state of the device with an assignment or instruction such as:
   - [:=] assignment (non-retentive assignment)
   - Output Energize (OTE) instruction in a subroutine

During the last scan of each step, the Automatic reset option:

- executes P and P0 actions according to their logic conditions
- clears tags to the left of [:=] assignments
- executes a **postscan** of embedded Structured Text
- executes a postscan of any subroutine that an action calls via a Jump to Subroutine (JSR) instruction
- resets any nested SFC (SFC that an action calls as a subroutine)

IMPORTANT The postscan of an action actually occurs when the action goes from active to inactive. Depending on the qualifier of the action, the postscan could occur before or after the last scan of the step.

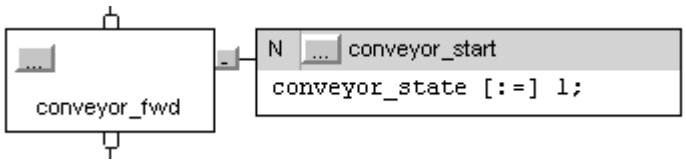
As a general rule, the postscan executes instructions as if all conditions are false. For example, the Output Energize (OTE) instruction clears its data during postscan.

Some instructions do not follow the general rule during postscan. For a description of how a specific instruction executes during postscan, see:

- *Logix5000 Controllers General Instructions Reference Manual*, publication 1756-RM003
- *Logix5000 Controllers Process and Drives Instructions Reference Manual*, publication 1756-RM006
- *Logix5000 Controllers Motion Instruction Set Reference Manual*, publication 1756-RM007

Here is an example that uses a non-retentive assignment to control a conveyor. It turns on a conveyor at the start of a step and automatically turns off the conveyor when the step is done.

EXAMPLE Automatically Clear Data



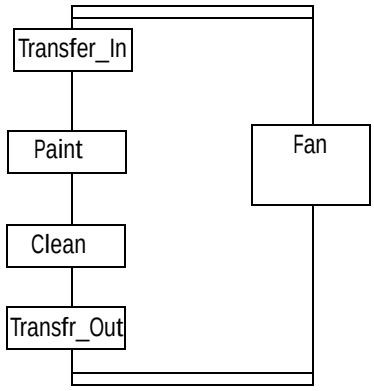
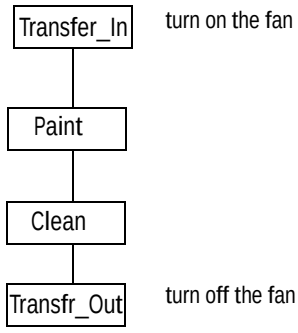
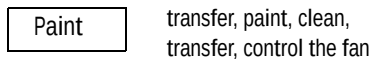
This action turns on the conveyor. When conveyor_state turns on, the conveyor turns on.

When the SFC leaves the step, conveyor_state turns off. This turns off the conveyor.

Keep Something On From Step-to-Step

How Do You Want to Control the Device?

To provide bumpless control of a device during more than one time or phase (step), do one of the following:

Option:	Example:
Use a Simultaneous Branch Make a separate step that controls the device.	
Store and Reset an Action Note the step that turns on the device and the step that turns off the device. Later, define a Stored and Reset Action pair to control the device.	
Use One Large Step Use one large step that contains all the actions that occur while the device is on.	

Use a Simultaneous Branch

A simple way to control a device or devices during one or more steps is to create a separate step for the devices. Then use a simultaneous branch to execute the step during the rest of the process.

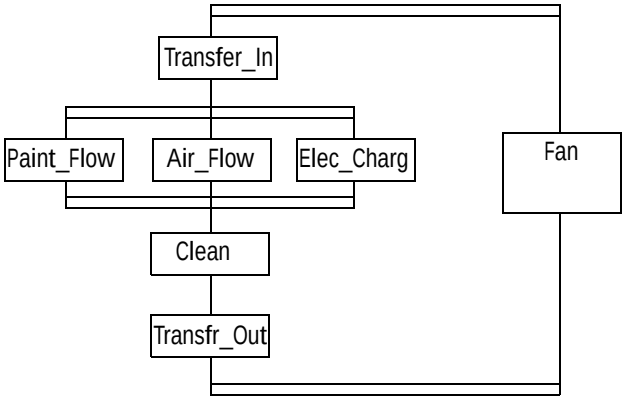
EXAMPLE

A paint operation:

1. Transfers the product into the paint shop.
2. Paints the product using 3 separate paint guns.
3. Cleans the guns.
4. Transfers the product to the paint ovens.

During the entire process, the system must control the shop fans.

Solution:



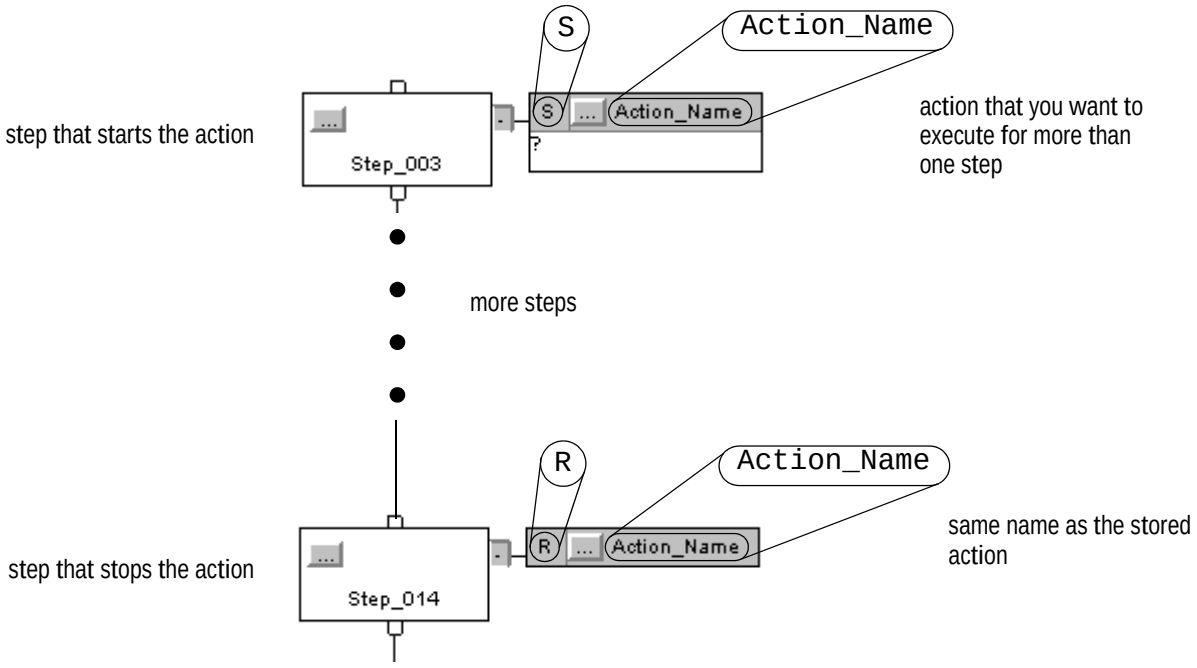
Store and Reset an Action

Typically, an action turns off (stops executing) when the SFC goes to the next step. To keep a device on from step to step without a bump, store the action that controls the device:

1. In the step that turns on the device, assign a stored qualifier to the action that controls the device. For a list of stored qualifiers, see Table 4.1 on page 4-140.

2. In the step that turns off the device, use a Reset action.

This figure shows the use of a stored action.

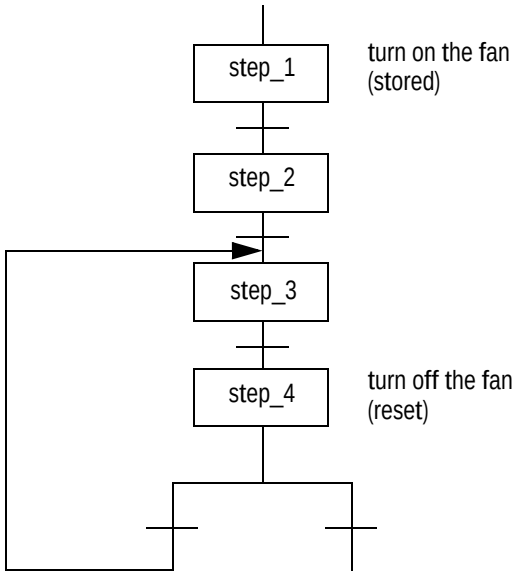


When the SFC leaves the step that stores the action, RSLogix 5000 software continues to show the stored action as active. (By default, a green border displays around the action.) This lets you know that the SFC is executing the logic of that action.

To use a stored action, follow these guidelines:

- The Reset action only turns off the stored action. It does not automatically turn off the devices of the action. To turn off the device, follow the Reset action with another action that turns off the device. Or use the Automatic reset option described on page 4-153.
- Before the SFC reaches a stop element, reset any stored actions that you do not want to execute at the stop. An active stored action remains active even if the SFC reaches a stop.
- Use caution when you jump in between a step that stores an action and a step that resets the action. Once you reset an action, it only starts when you execute the step that stores the action.

In this example, steps 1 - 4 require a fan to be on. At the end of step_4, the fan is reset (turned off). When the SFC jumps back to step_3, the fan remains off.



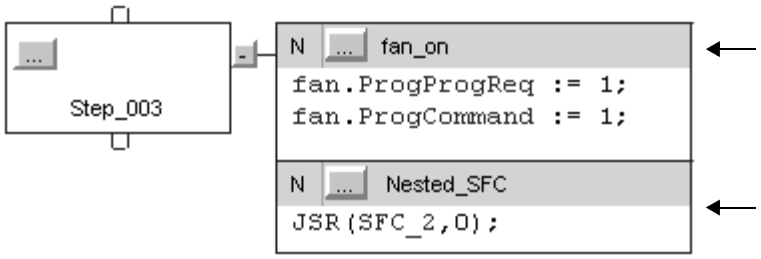
To turn the fan back on, the SFC has to jump back to step_1.

Use One Large Step

If you use one large step for multiple functions, then use additional logic to sequence the functions. One option is to nest an SFC within the large step.

In this example, a step turns on a fan and then calls another SFC. The nested SFC sequences the remaining functions of the step. The fan stays on throughout the steps of the nested SFC.

EXAMPLE Use a Large Step



This action turns on a fan:

- fan.ProgProgReq lets the SFC command the state of the fan.
- fan.ProgCommand turns on the fan.

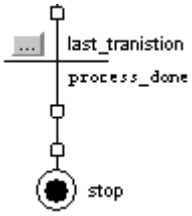
This action calls another SFC. The SFC sequences the remaining functions of the step.

For additional information on how to nest an SFC, see "Nest an SFC" on page 4-161.

End the SFC

Once an SFC completes its last step, it does not automatically restart at the first step. You must tell the SFC what to do when it finishes the last step.

To:	Do this:
automatically loop back to an earlier step	Wire the last transition to the top of the step to which you want to go. See "Wire to a Previous Step"on page 4-134.
stop and wait for a command to restart	Use a Stop Element. See "Use a Stop Element" on page 4-158.



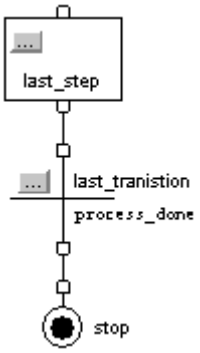
Use a Stop Element

The stop element lets you stop the execution of an entire SFC or a path of a simultaneous branch and wait to restart. When an SFC reaches a stop element:

- The X bit of the stop element turns on. This signals that the SFC is at the stop element.
- Stored actions remain active.
- Execution stops for part or all of the SFC:

If the stop element is at the end of a:	Then:
sequence	entire SFC stops
selection branch	
path within a simultaneous branch	only that path stops while the rest of the SFC continues to execute.

EXAMPLE Use a Stop Element



When the SFC reaches last_step and process_done is true, the execution of the SFC stops.

Restart (Reset) the SFC

Once at the stop element, you have several options to restart the SFC:

If the SFC is:	And the *Last Scan of Active Steps* option is:	Then:
nested (i.e., another SFC calls this SFC as a subroutine)	Automatic reset	At the end of the step that calls the nested SFC, the nested SFC automatically resets: • The nested SFC resets to the initial step. • The X bit of the stop element in the nested SFC clears to zero.
	Programmatic reset Don't scan	1. Use an SFC Reset (SFR) instruction to restart the SFC at the required step. 2. Use logic to clear the X bit of the stop element.
NOT nested (i.e., no SFC calls this SFC as a subroutine)	→	1. Use an SFC Reset (SFR) instruction to restart the SFC at the required step. 2. Use logic to clear the X bit of the stop element.

This example shows the use of the SFC Reset (SFR) instruction to restart the SFC and clear the X bit of the stop element.

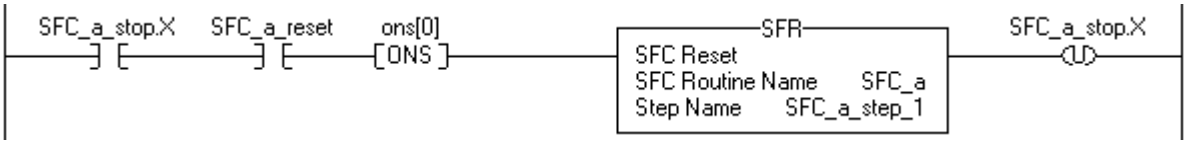
EXAMPLE

Restart (Reset) the SFC

If SFC_a_stop.X = on (SFC_a is at the stop) and SFC_a_reset = on (time to reset the SFC) then for one scan (ons[0] = on):

Reset SFC_a to SFC_a_Step_1

SFC_a_stop.X = 0



SFC_STOP Structure

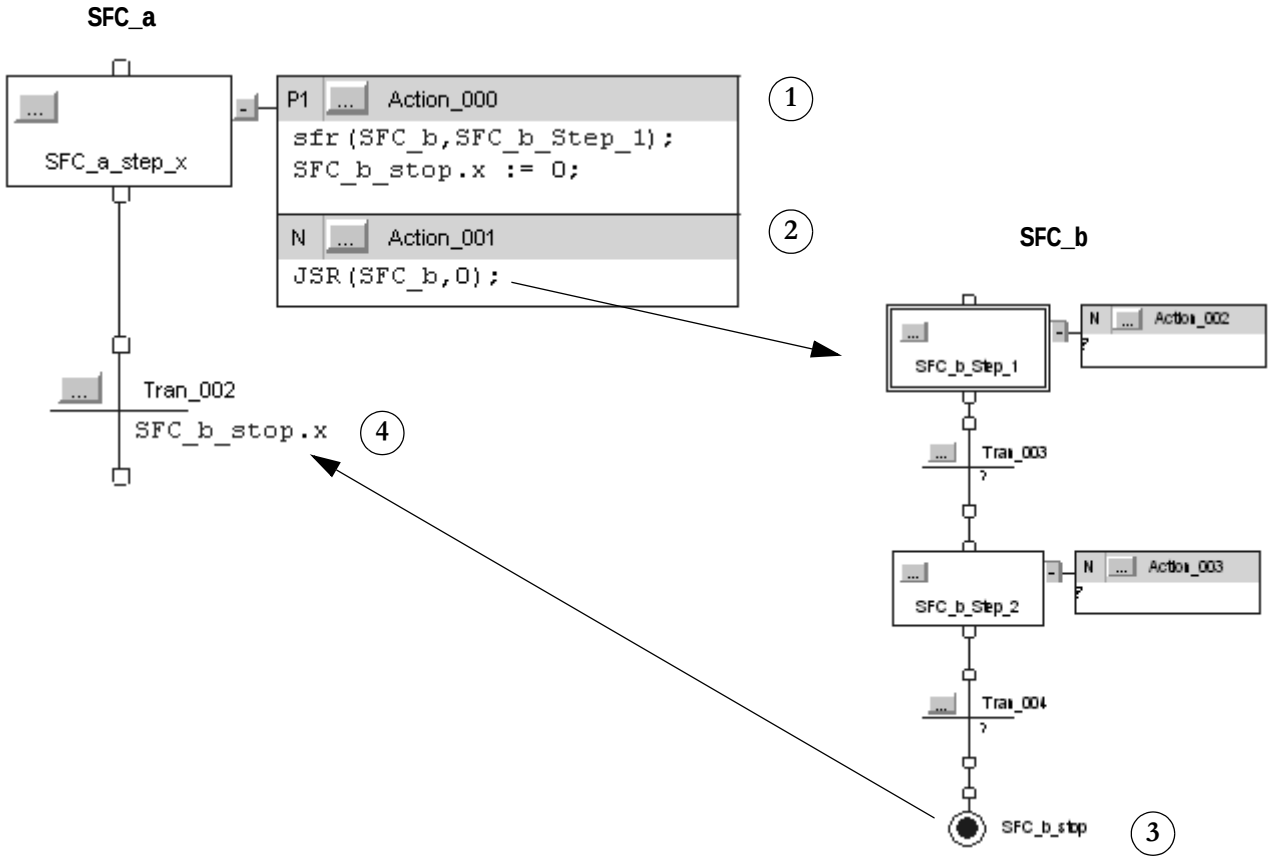
Each stop uses a tag to provide information about the stop element:

If you want to:	Then check or set this member:	Data type:	Details:
determine when the SFC is at the stop	X	BOOL	• When the SFC reaches the stop, the X bit turns on. • The X bit clears if you configure the SFCs to restart at the initial step and the controller changes from program to run mode. • In a nested SFC, the X bit also clears if you configure the SFCs for automatic reset and the SFC leaves the step that calls the nested SFC.
determine the target of an SFC Reset (SFR) instruction	Reset	BOOL	An SFC Reset (SFR) instruction resets the SFC to a step or stop that the instruction specifies. • The Reset bit indicates to which step or stop the SFC will go to begin executing again. • Once the SFC executes, the Reset bit clears.
determine how many times a stop has become active	Count	DINT	This is not a count of scans of the stop. • The count increments each time the stop becomes active. • It increments again only after the stop goes inactive and then active again. • The count resets only if you configure the SFC to restart at the initial step. With that configuration, it resets when the controller changes from program mode to run mode.
use one tag for the various status bits of this stop	Status	DINT	**For this member:** / **Use this bit:** Reset / 22 X / 31

Nest an SFC

One method for organizing your project is to create one SFC that provides a high-level view of your process. Each step of that SFC calls another SFC that performs the detailed procedures of the step (nested SFC).

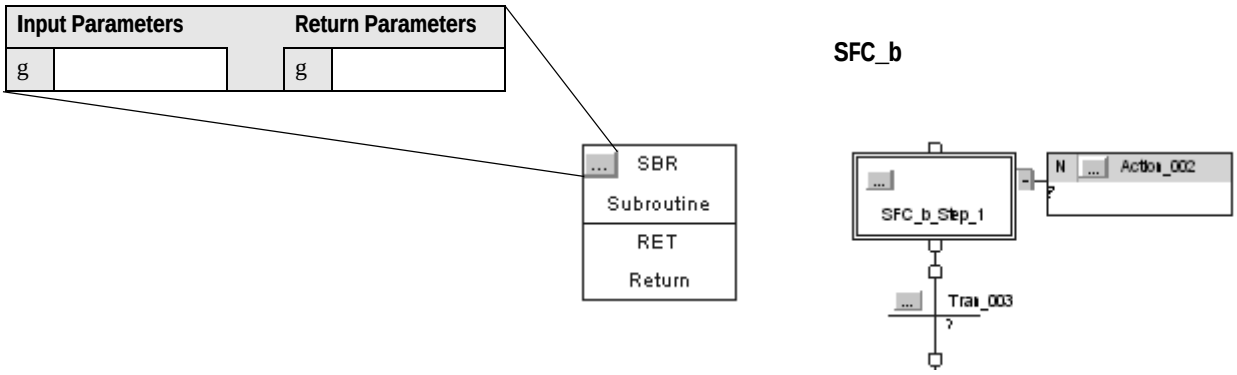
This figure shows one way to nest an SFC. In this method, the last scan option of the SFC is configured for either Programmatic reset or Don't scan. If you configure the SFC for Automatic reset, then step 1 is unnecessary.



1. Reset the nested SFC:
   - The SFR instruction restarts SFC_b at SFC_b_Step_1. Each time SFC_a leaves this step and then returns, you have to reset SFC_b.
   - The action also clears the X bit of the stop element.
2. Call SFC_b.
3. Stop SFC_b. This sets the X bit of the stop element.
4. Use the X bit of the stop element to signal that SFC_b is done and it is time to go to the next step.

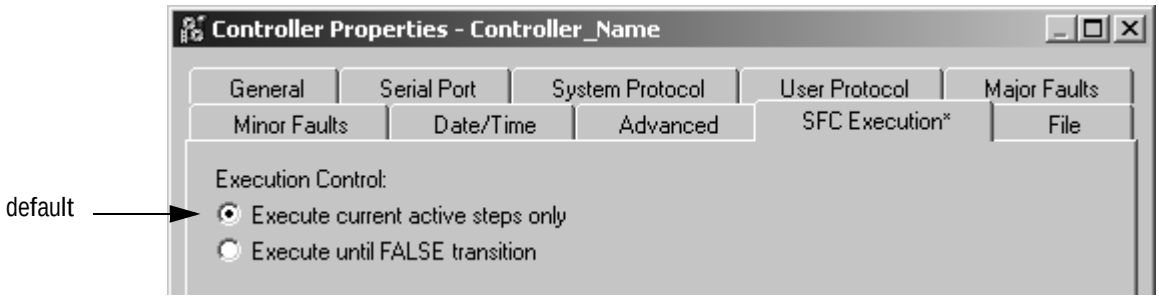
Pass Parameters

To pass parameters to or from an SFC, place a Subroutine/Return element in the SFC.



Configure When to Return to the OS/JSR

By default, an SFC executes a step or group of simultaneous steps and then returns to the operating system (OS) or the calling routine (JSR).



You have the option of letting the SFC execute until it reaches a false transition. If several transitions are true at the same time, this option reduces the time to get to the desired step.

Use the Execute until FALSE transition option only when:

1. You don't have to update JSR parameters before each step. Parameters update only when the SFC returns to the JSR. See "Pass Parameters" on page 4-162.

2. A false transition occurs within the watchdog timer for the task. If the time that it takes to return to a JSR and complete the rest of the task is greater than the watchdog timer, a major fault occurs.

Pause or Reset an SFC

Two optional instructions are available that give you further control over the execution of your SFC:

If you want to:	Then use this instruction:
pause an SFC	Pause SFC (SFP)
reset an SFC to a specific step or stop	Reset SFC (SFR)

Both instructions are available in the ladder logic and Structured Text programming languages.

Execution Diagrams

The following diagrams show the execution of an SFC with different organizations of steps or different selections of execution options.

For a diagram of the:	Page
Execution of a Sequence	164
Execution of a Simultaneous Branch	165
Execution of a Selection Branch	166
When parameters enter and exit an SFC	166
Options for Execution Control	167

Figure 4.5 Execution of a Sequence

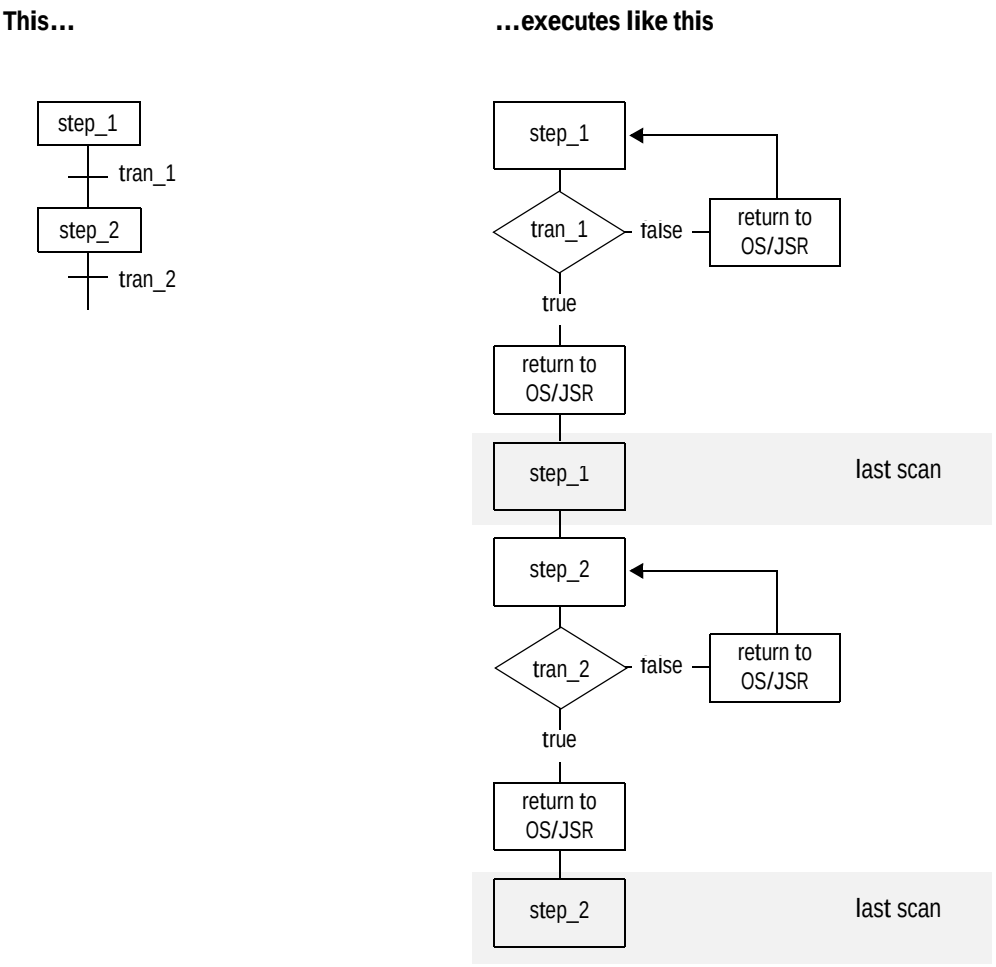
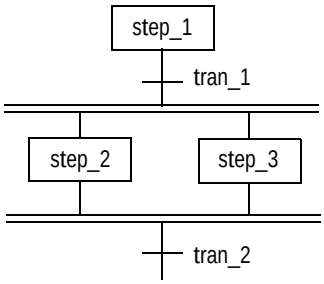


Figure 4.6 Execution of a Simultaneous Branch

This…



…executes like this

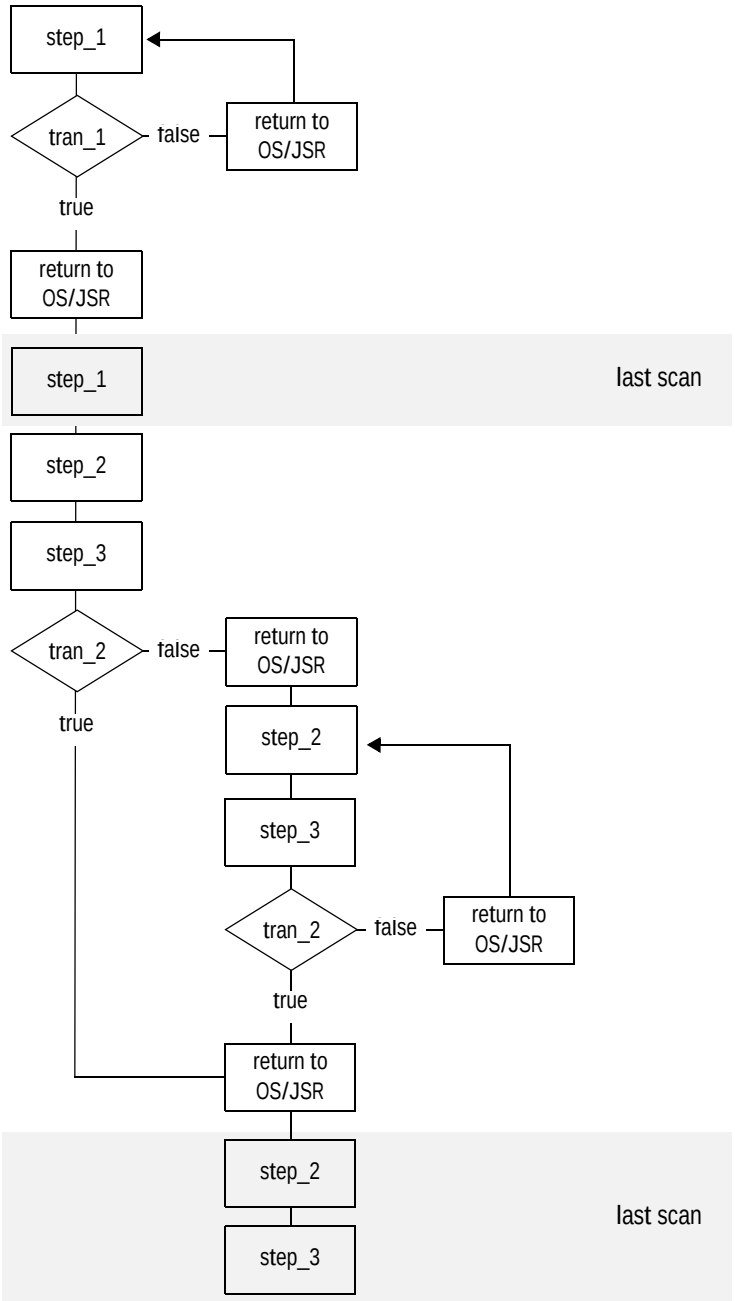


Figure 4.7 Execution of a Selection Branch

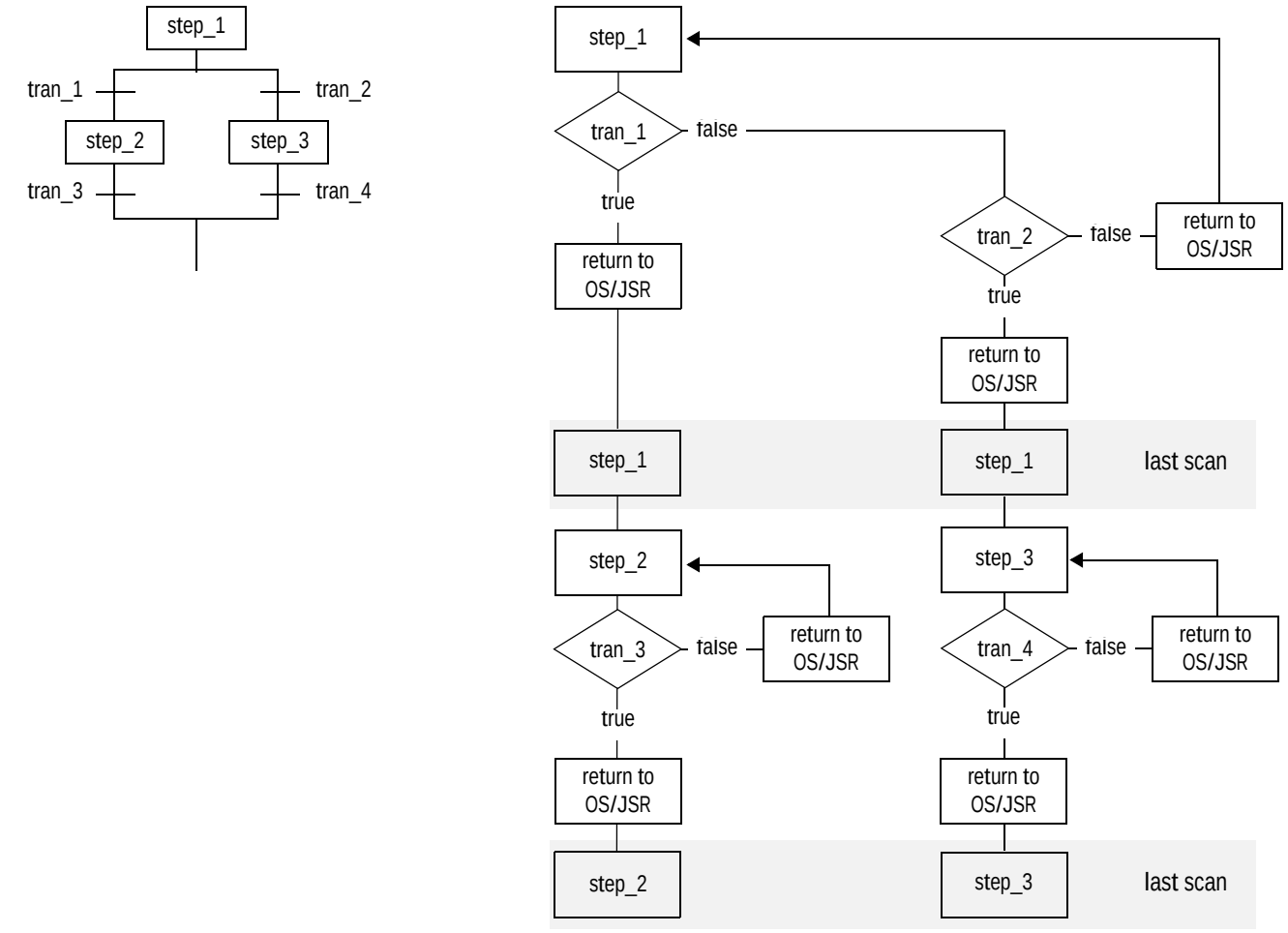


Figure 4.8 When parameters enter and exit an SFC

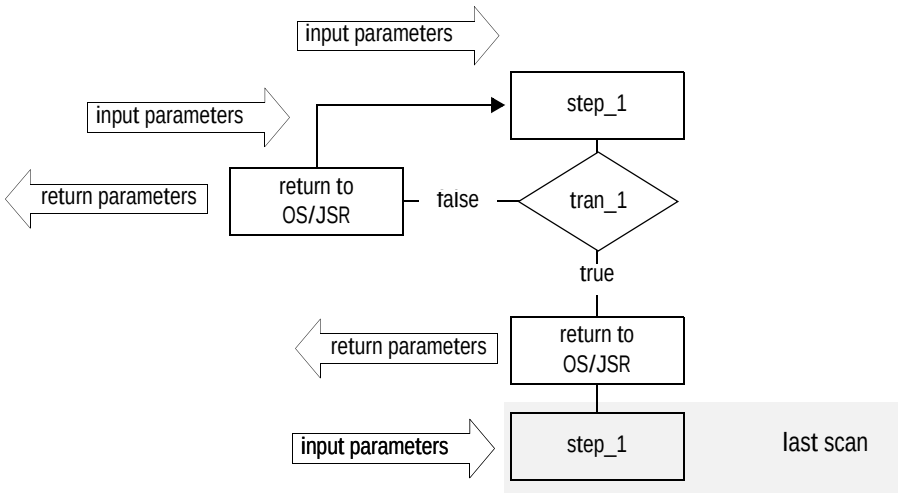
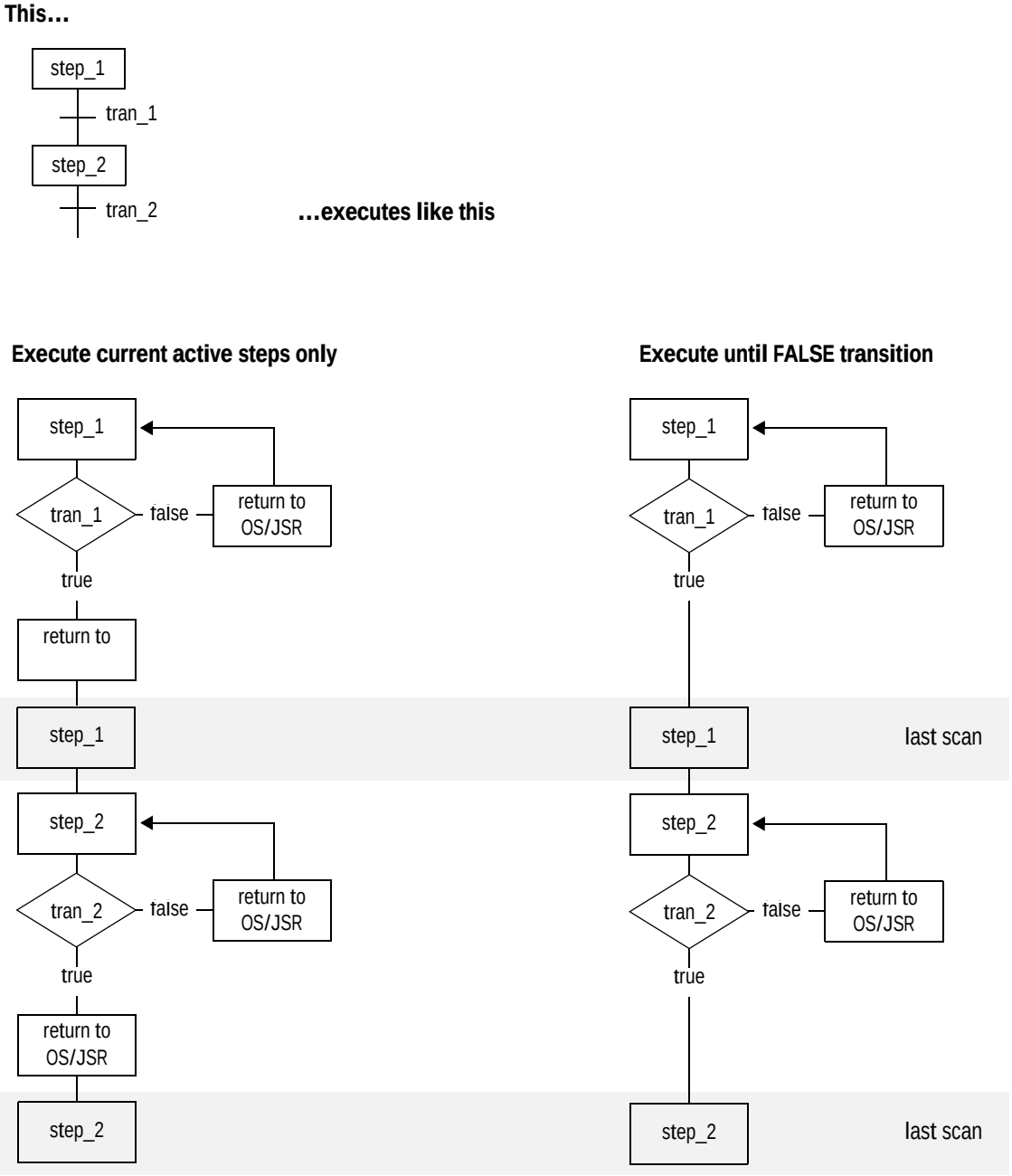


Figure 4.9 Options for Execution Control



Notes:

Programming a Sequential Function Chart

When to Use This Chapter

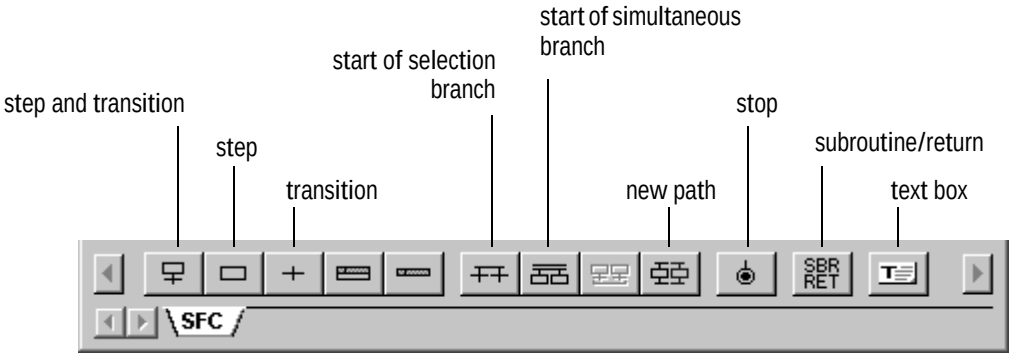
Use this chapter to enter a **Sequential Function Chart** (SFC) into RSLogix 5000 software. Enter the SFC as you design it. Or first design the SFC and then enter it. To design the SFC, see "Designing a Sequential Function Chart" on page 121.

To program a sequential function chart routine:

Topic	Page
Add an SFC Element	169
Create a Simultaneous Branch	171
Create a Selection Branch	173
Set the Priorities of a Selection Branch	174
Return to a Previous Step	175
Configure a Step	176
Program a Transition	178
Add an Action	180
Configure an Action	180
Program an Action	182
Assign the Execution Order of Actions	184
Document an SFC	184
Configure the Execution of the SFC	188
Verify the Routine	189

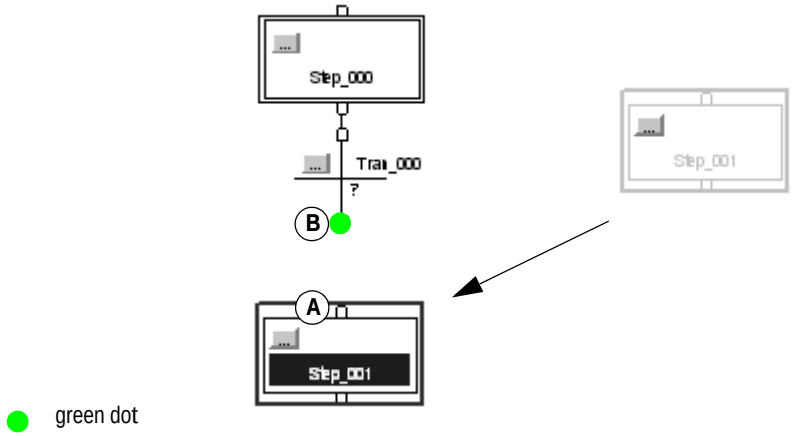
Add an SFC Element

To add SFC elements, use the SFC toolbar.



Add and Manually Connect Elements

1. On the SFC toolbar, click the button for the item that you want to add.
2. Drag the element to the required location on the SFC.



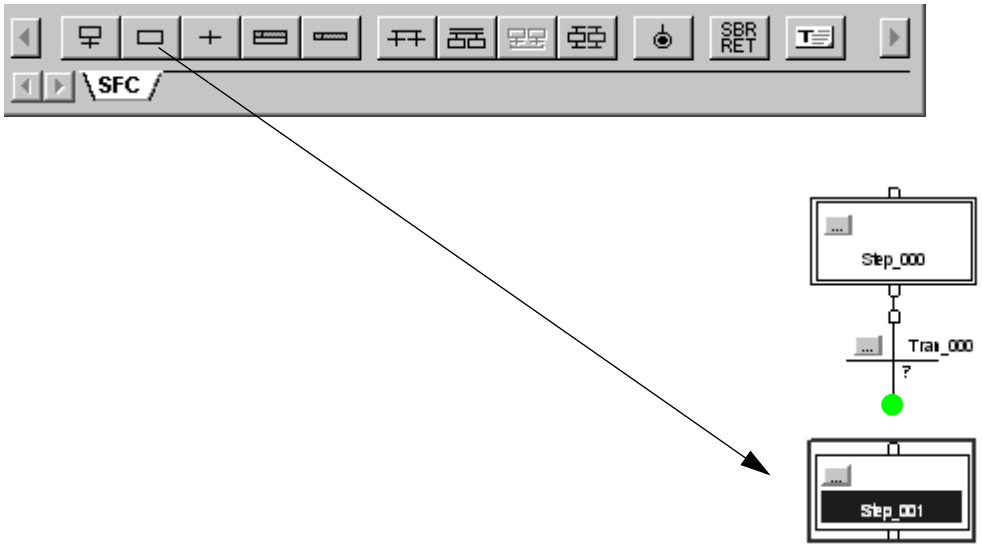
3. To wire (connect) two elements together, click a pin on one of the elements (A) and then click the pin on the other element (B). A green dot shows a valid connection point.

Add and Automatically Connect Elements

1. Select (click) the element to which you want to connect a new element.
2. With the element still selected, click the toolbar button for the next element.

Drag and Drop Elements

From the SFC toolbar, drag the button for the required element to the desired connection point on the SFC. A green dot shows a valid connection point.

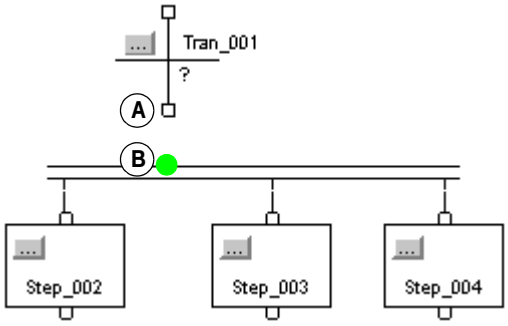


● green dot

Create a Simultaneous Branch

Start a Simultaneous Branch

1. On the SFC toolbar, click the  button. Drag the new branch to the desired location.
2. To add a path to the branch, select (click) the first step of the path that is to the left of where you want to add the new path. Click 



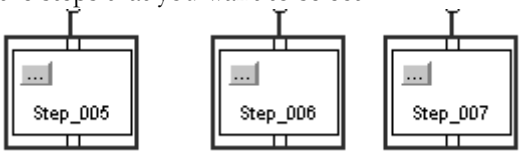
● green dot

3. To wire the simultaneous branch to the preceding transition, click the bottom pin of the transition (A) and then click the horizontal line of the branch (B). A green dot shows a valid connection point.

End a Simultaneous Branch

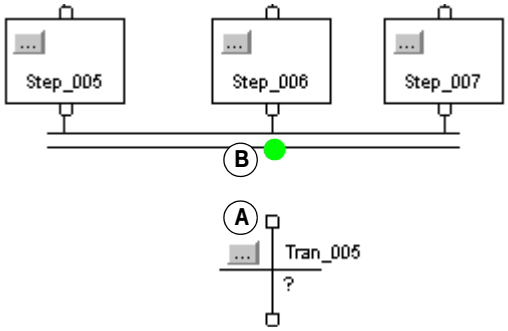
1. Select the last step of each path in the branch. To select the steps, you can either:

   - Click and drag the pointer around the steps that you want to select.
   - Click the first step. Then press and hold [Shift] and click the rest of the steps that you want to select.



2. On the SFC toolbar, click 

3. Add the transition that follows the simultaneous branch.



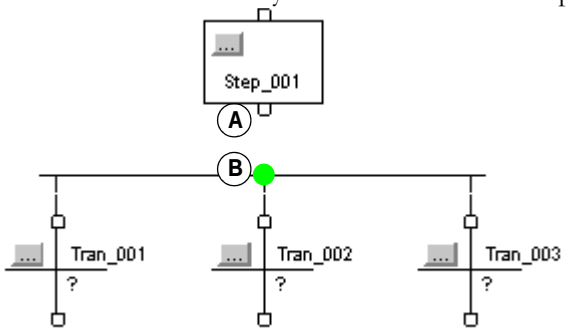
4. To wire the simultaneous branch to the transition, click the top pin of the transition (A) and then click the horizontal line of the branch (B). A green dot shows a valid connection point.

Create a Selection Branch

Start a Selection Branch

1. On the SFC toolbar, click the  button. Then drag the new branch to the desired location.

2. To add a path to the branch, select (click) the first transition of the path that is to the left of where you want to add the new path. Click 



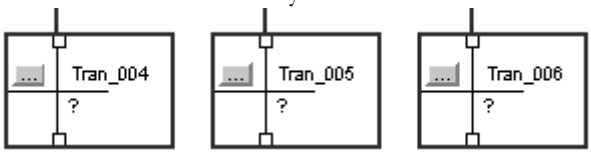
green dot

3. To wire the selection branch to the preceding step, click the bottom pin of the step (A) and then click the horizontal line of the branch (B). A green dot shows a valid connection point.

End a Selection Branch

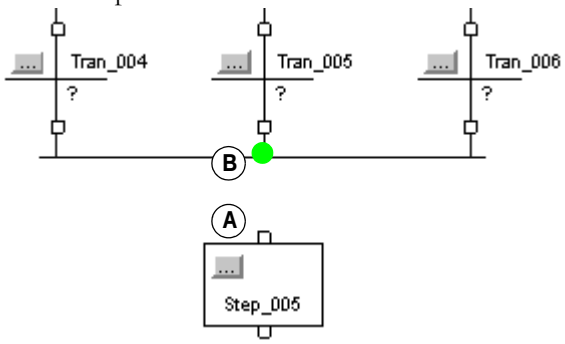
1. Select the last transition of each path in the branch. To select the transitions, you can either:

   - Click and drag the pointer around the transitions that you want to select.
   - Click the first transition. Then press and hold [Shift] and click the rest of the transitions that you want to select.



2. On the SFC toolbar, click 

3. Add the step that follows the selection branch.



green dot

4. To wire the selection branch to the step, click the top pin of the step (A) and then click the horizontal line of the branch (B). A green dot shows a valid connection point.

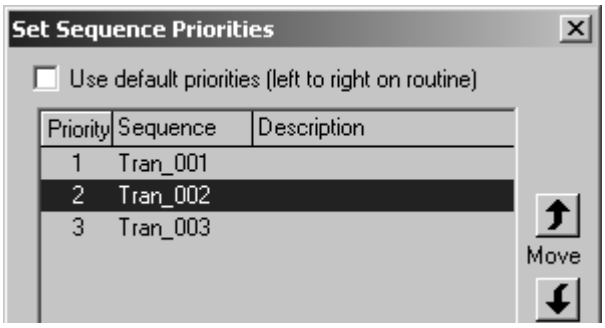
Set the Priorities of a Selection Branch

By default, the SFC checks the transitions that start a selection branch from left to right. If you want to check a different transition first, assign a priority to each path of the selection branch. For example, it is a good practice to check for error conditions first. Then check for normal conditions.

To assign priorities to a selection branch:

1. Right-click the horizontal line that starts the branch and select Set Sequence Priorities.

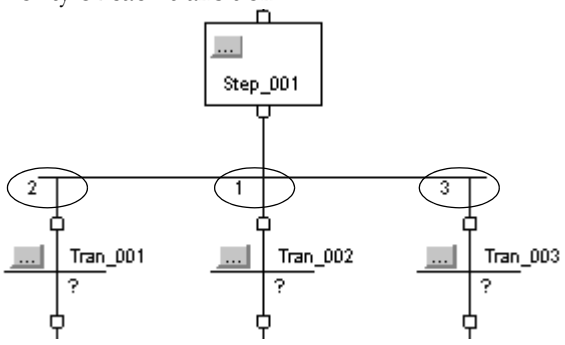
2. Clear (uncheck) the Use default priorities check box and select a transition.



Use the Move buttons to raise or lower the priority of the transition.

3. When all the transitions have the desired priority, click OK

When you clear (uncheck) the Use default priorities check box, numbers show the priority of each transition.



Return to a Previous Step

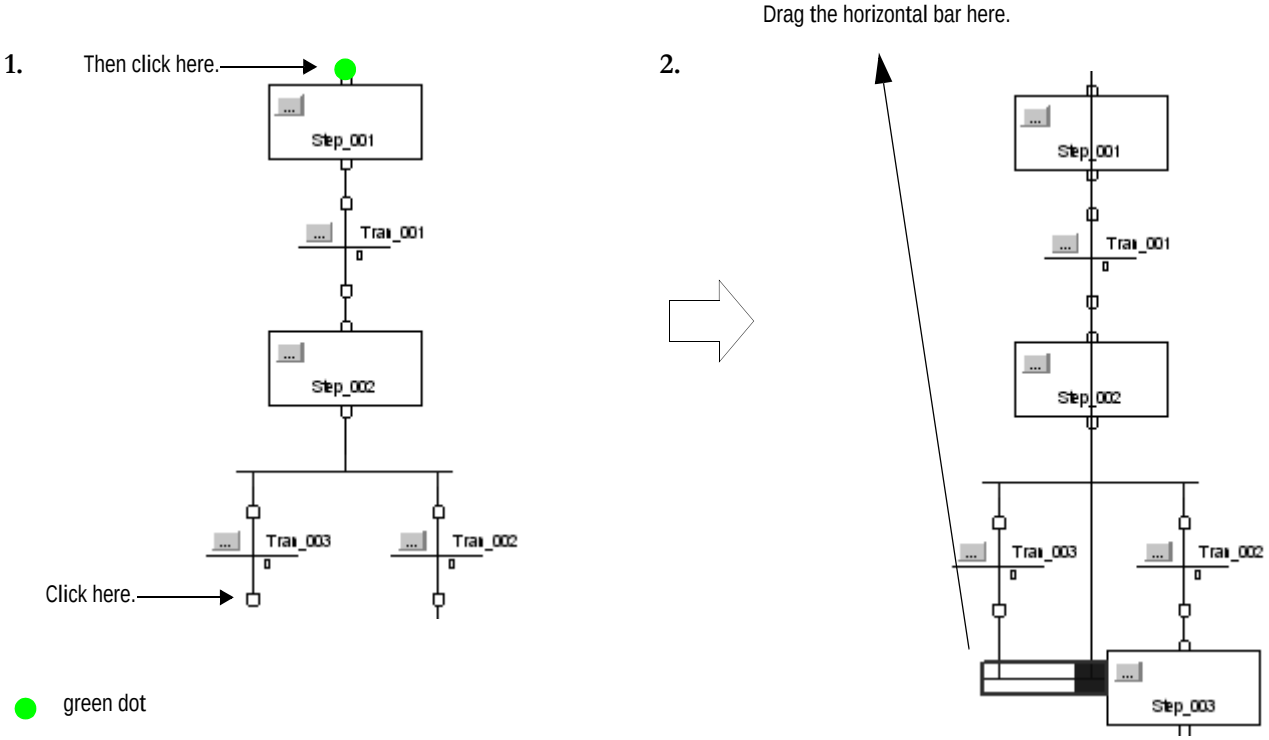
Connect a Wire to the Step

1. Click the lower pin of the transition that signals the jump. Then click the top pin of the step to which you want to go. A green dot shows a valid connection point.

   Typically, the resulting connection orients itself along the center of the flowchart and is hard to see.

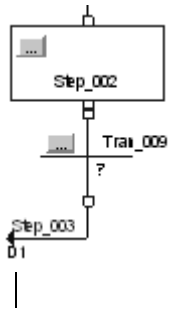
2. To make the jump easier to read, drag its horizontal bar above the step to which the jump goes. You may also have to reposition some of the SFC elements.

For example, to go to Step_001 from Tran_003:



Hide a Wire

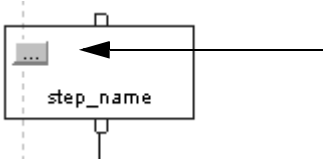
If a wire gets in the way of other parts of your SFC, hide the wire to make the SFC easier to read. To hide a wire, right-click the wire and select Hide Wire.



To see the SFC element to which the wire goes, click the grid location on the wire.

Configure a Step

Assign the Preset Time for a Step



1. Click the ... button of the step.



2. On the General tab, type the time for the step, in milliseconds.

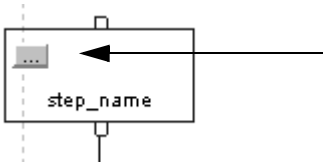
3. Click OK

When the step is active for the preset time (Timer = Preset), the DN bit of the step turns on.

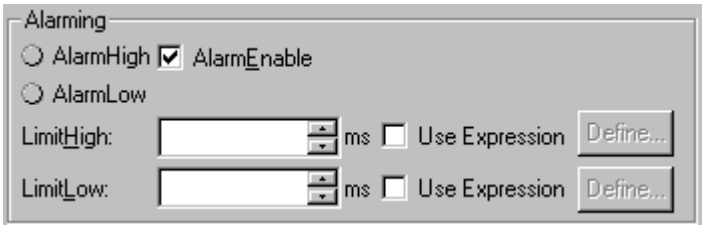
To calculate the preset time for a step at runtime, see "Use an Expression to Calculate a Time" on page 177.

Configure Alarms for a Step

To turn on an alarm if a step executes too long or not long enough:



1. Click the ... button of the step.
2. Check the AlarmEnable check box.



Type the time for the high alarm and low alarm, in milliseconds.

3. Click OK

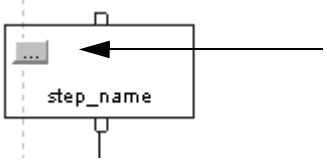
To calculate the time for an alarm at runtime, see "Use an Expression to Calculate a Time" on page 177.

Use an Expression to Calculate a Time

To calculate a time based on tags in your project, enter the time as a **numeric expression**. You can use an expression to calculate:

- Preset
- LimitHigh
- LimitLow

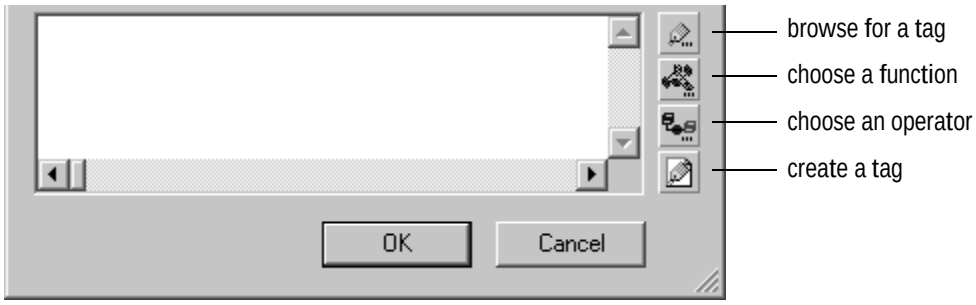
To enter a time as an expression:



1. Click the ... button of the step.
2. Check the Use Expression check box.



3. Click the Define button and enter an expression.



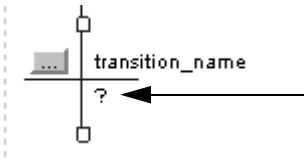
Type a **numeric expression** that defines the time.

- Use the buttons alongside the dialog box to help you complete the expression.
- For information on numeric expressions, see "Expressions" on page 194.

4. Click OK

5. To close the Step Properties dialog box, click OK

Program a Transition



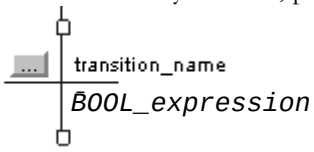
Enter a BOOL Expression

The simplest way to program the transition is to enter the conditions as a **BOOL expression** in Structured Text. For information on BOOL expressions, see "Expressions" on page 194.

1. Double-click the text area of the transition.

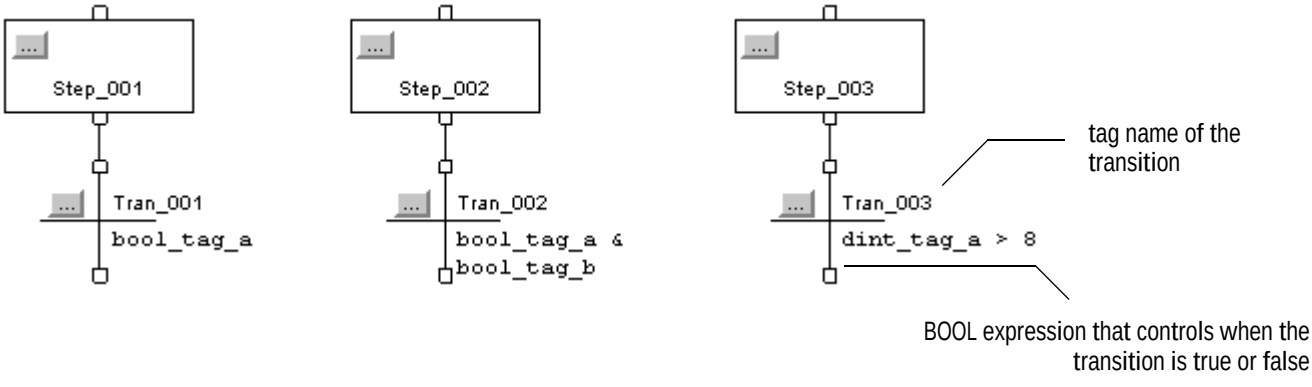
2. Type the BOOL expression that determines when the transition is true or false.

3. To close the text entry window, press [Ctrl] + [Enter].

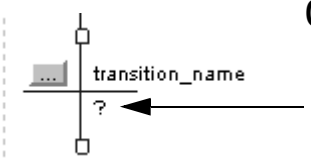


This example shows three transitions that use a BOOL expression.

EXAMPLE Enter a BOOL Expression



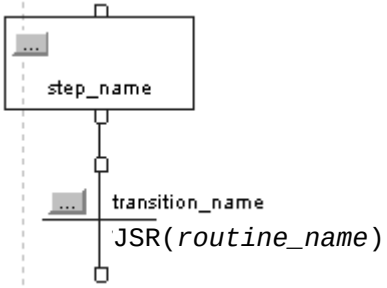
Call a Subroutine



1. Right-click the transition and select Set JSR.
2. Select the routine that contains the logic for the transition.

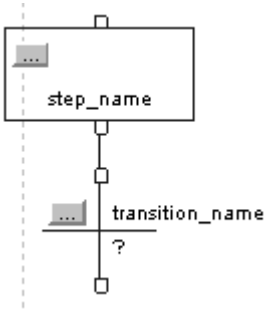


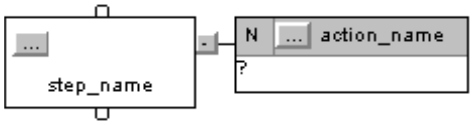
3. Click OK



Add an Action

To add an action to a step, right-click the step in which the action executes and select Add Action.

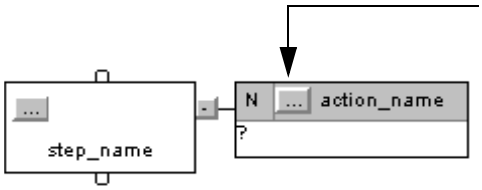




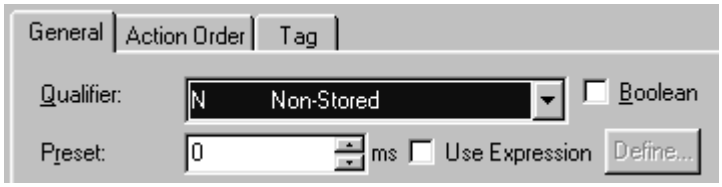
Configure an Action

Change the Qualifier of an Action

A qualifier determines when an action starts and stops. The default qualifier is N Non-Stored. The action starts when the step is activated and stops when the step is deactivated. For more information, see "Choose a Qualifier for an Action" on page 139.



1. Click the [...] button of the action.
2. On the General tab, select the qualifier for the action.



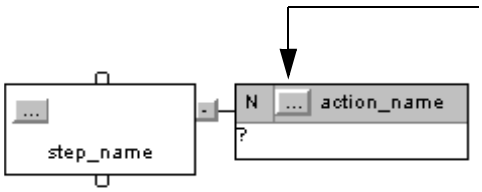
   If you chose a timed qualifier, type the time limit or delay for the action, in milliseconds. Timed qualifiers include:

   - L Time Limited
   - SL Stored and Time Limited
   - D Time Delayed
   - DS Delayed and Stored
   - SD Stored and Time Delayed

3. Click OK

Calculate a Preset Time at Runtime

To calculate a preset value based on tags in your project, enter the value as a **numeric expression**.



1. Click the ... button of the action.
2. Check the Use Expression check box.



3. Click the Define button.



Type a **numeric expression** that defines the preset time.

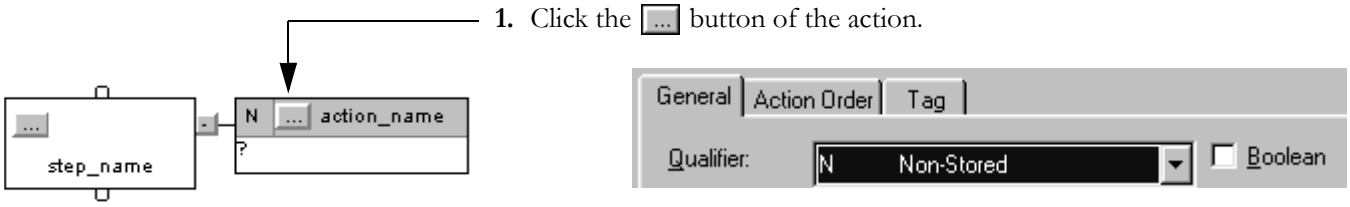
- Use the buttons alongside the dialog box to help you complete the expression.
- For information on numeric expressions, see "Expressions" on page 194.

4. Click OK

5. To close the Action Properties dialog box, click OK

Mark an Action as a Boolean Action

Use a boolean action to only set a bit when the action executes. For more information, see "Use a Boolean Action" on page 137.



1. Click the ... button of the action.
2. Check the Boolean check box.
3. Click OK

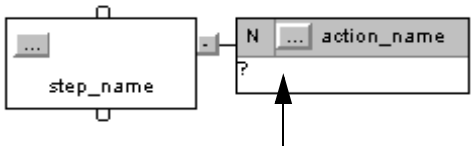
Program an Action

To program an action, you can:

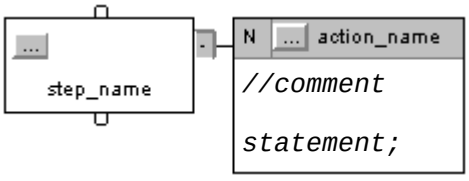
- Enter Structured Text
- Call a Subroutine

Enter Structured Text

The easiest way to program an action is to write the logic as Structured Text within the body of the action. When the action turns on, the controller executes the Structured Text.



1. Double-click the text area of the action.
2. Type the required Structured Text.
3. To close the text entry window, press [Ctrl] + [Enter].

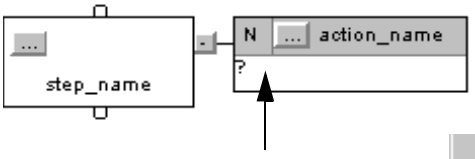


For information on Structured Text:

For this Structured Text information:	See:
general information about assignments, operators, functions, instructions, or comments	"Programming Structured Text" on page 191
information about a specific instruction	• *Logix5000 Controllers General Instructions Reference Manual*, publication 1756-RM003 • *Logix5000 Controllers Process and Drives Instructions Reference Manual*, publication 1756-RM006 • *Logix5000 Controllers Motion Instruction Set Reference Manual*, publication 1756-RM007

Call a Subroutine

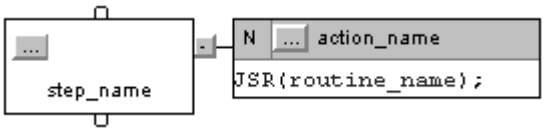
Use a Jump to Subroutine (JSR) instruction to execute a subroutine when the action is active.



1. In the SFC, right-click the text entry area of the action and select Set JSR.

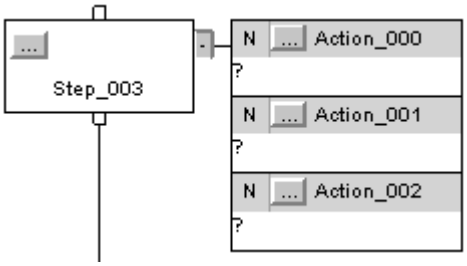


2. Choose the routine that you want to call.
3. To pass a parameter to the routine, click an empty Input Parameters text box. Then use the down arrow to select the tag that contains the parameter.
4. To receive a parameter from the routine, click an empty Return Parameters text box. Then use the down arrow to select the tag in which to store the parameter from the routine.
5. Click OK



Assign the Execution Order of Actions

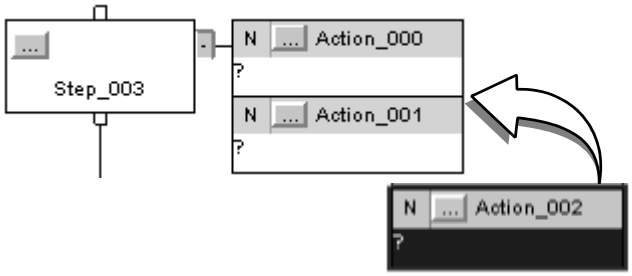
Actions execute in the order in which they appear.



When Step_003 is active, its actions execute in this order:

1. Action_000
2. Action_001
3. Action_002

To change the order in which an action executes, drag the action to the desired location in the sequence. A green bar shows a valid placement location.



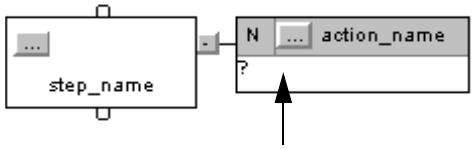
Document an SFC

To document an SFC, you can:

To document this:	And you want to:	Do this:
general information about the SFC	→	Add a Text Box
step	→	Add a Text Box -or- Add a Tag Description
transition	download the documentation to the controller	Add Structured Text Comments
	have the option of showing or hiding the documentation	Add a Text Box -or- Add a Tag Description
	position the documentation anywhere in the SFC	
action	download the documentation to the controller	Add Structured Text Comments
stop	→	Add a Text Box -or- Add a Tag Description
other element (e.g., selection branch)	→	

Add Structured Text Comments

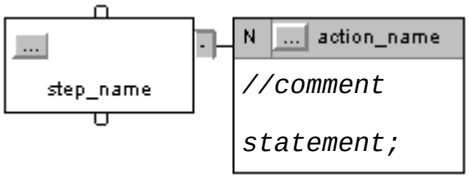
Comments embedded in the ST section of an action are downloaded into controller memory and are available for upload. To enter comments:



1. Double-click the text area of the action.
2. Type the comments.

To add a comment:	Use one of these formats:
on a single line	`//comment`
at the end of a line of Structured Text	`(*comment*)` `/*comment*/`
within a line of Structured Text	`(*comment*)` `/*comment*/`
that spans more than one line	`(*start of comment . . . end of comment*)` `/*start of comment . . . end of comment*/`

3. To close the text entry window, press [Ctrl] + [Enter].



Add a Tag Description

1. Click the [...] button of the element.
2. Click the Tag tab and type the description for the element.

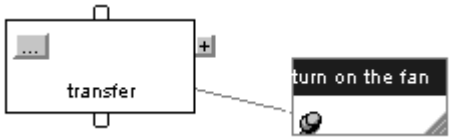


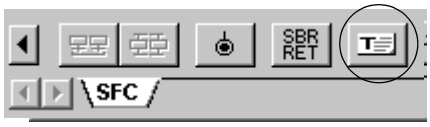
3. Click [OK]
4. Drag the description box to the desired location on the SFC.

Add a Text Box

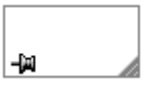
A text box lets you add notes that clarify the function of an SFC element (step, transition, stop, etc.). Text boxes are only stored in the offline, ACD project file. Text boxes are not downloaded into controller memory.

Or use a text box to capture information that you will use later on. For example:





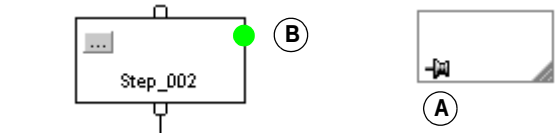
1. Click [T]

   A text box appears. 

2. Drag the text box to a location near the element to which it applies.
3. Double-click the text box and type the note. Then press [Ctrl] + [Enter].

4. As you move the element on the SFC, what do you want the text box to do?

If you the text box to:	Then:
stay in the same spot	Stop. You are done.
move with the element to which it applies	Go to step 5.



green dot

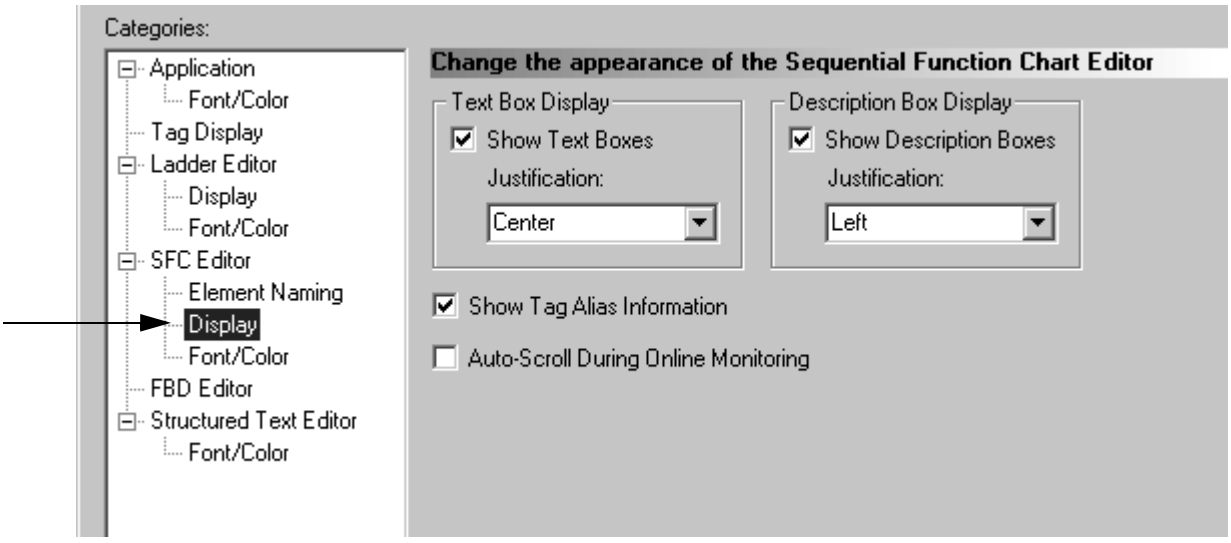
5. Click the pin symbol in the text box and then click the SFC element to which you want to attach the text box. A green dot shows a valid connection point.

Show or Hide Text Boxes or Tag Descriptions

You have the option of showing or hiding both text boxes and tag descriptions. If you select to show descriptions, the SFC window only shows the descriptions for steps, transitions, and stops (not actions).

1. From the Tools menu, select Options.

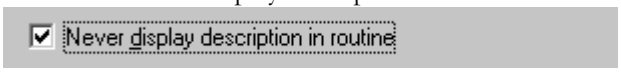
2. Under SFC Editor, select the Display category.



Hide an Individual Tag Description

To hide the description of a specific element while showing other descriptions:

1. Click the [...] button of the element whose description you want to hide.
2. Check the Never display description in routine check box.



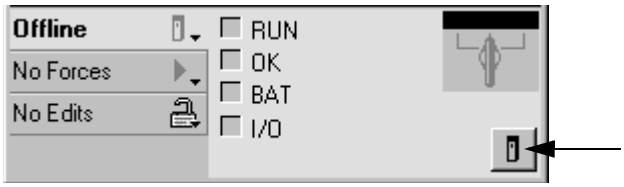
3. Click [OK]

Configure the Execution of the SFC

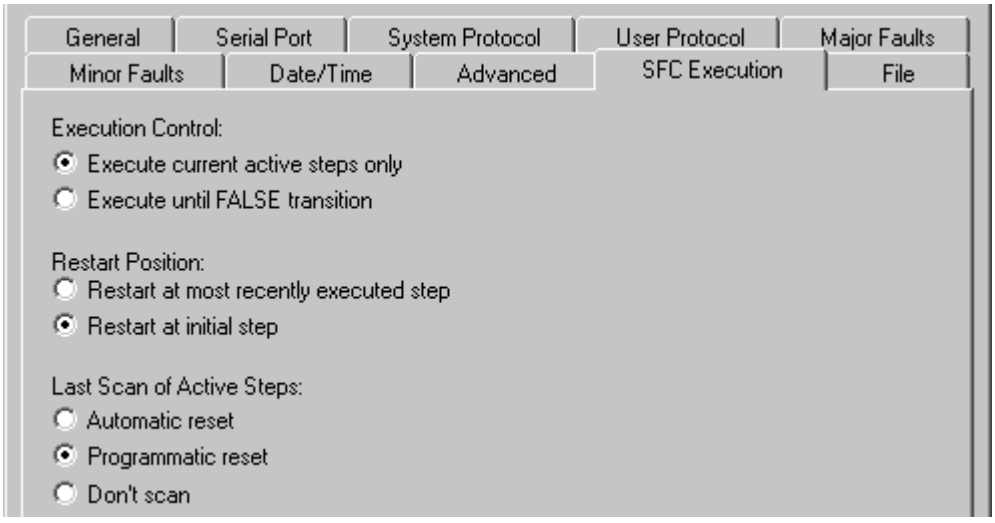
The SFC Execution tab of the controller properties lets you configure:

- what to do when a transition is true
- where to start after a transition to the Run mode or recovery from a power loss
- what to do on the last scan of a step

1. On the Online toolbar, click the controller properties button.



2. Select the SFC Execution tab.



3. Choose:
   - whether or not to return to the OS/JSR if a transition is true.
   - where to restart the SFC.

     The restart position applies when the controller loses power or leaves the Run or Remote Run mode. Select:

If you want to restart at the:	Select
Last step that was running	Restart at most recently executed step
Initial step	Select Restart at initial step

     The restart position isn't for major faults. After you clear a major fault, the SFC **always** restarts at the initial step.

   - what to do on the last scan of a step.

4. Click OK

Verify the Routine

As you program your routine, periodically verify your work:

1. In the top-most toolbar of the RSLogix 5000 window, click 
2. If any errors are listed at the bottom of the window:
   a. To go to the first error or warning, press [F4].
   b. Correct the error according to the description in the Results window.
   c. Go to step 1.
3. To close the Results window, press [Alt] + [1].

To check your SFC, you can:

- force transitions
- step through the SFC

For more information on these debug options, see chapter 15 Forcing Logic Elements.

Edit an SFC Online

Firmware revision 13 added support for editing SFCs online. When you transition the controller to test or un-test edits, the controller resets the SFC and starts execution at the initial step. If you edit an SFC online:

- Time when you test or untest edits to coincide with the SFC executing the initial step.
- Place Structured Text logic in subroutines to minimize the impact of online edits.
- Use an SFR instruction to programmatically shift SFC execution to the desired step.

Programming Structured Text

When to Use This Chapter

Use this chapter to write and enter Structured Text for a:

- Structured Text routine
- action of a sequential function chart (SFC)
- transition of sequential function chart (SFC)

Structured Text Syntax

Structured Text is a textual programming language that uses statements to define what to execute.

- Structured Text is not case sensitive.
- Use tabs and carriage returns (separate lines) to make your Structured Text easier to read. They have no effect on the execution of the Structured Text.

Structured Text can contain these components:

Term:	Definition:		Examples:
assignment (see page 6-192)	Use an assignment statement to assign values to tags. The := operator is the assignment operator. Terminate the assignment with a semi colon ";".		`tag := expression;`
expression (see page 194)	An expression is part of a complete assignment or construct statement. An expression evaluates to a number (numerical expression) or to a true or false state (BOOL expression). An expression contains:		
	tags	A named area of the memory where data is stored (BOOL, SINT,INT,DINT, REAL, string).	`value1`
	immediates	A constant value.	`4`
	operators	A symbol or mnemonic that specifies an operation within an expression.	`tag1 + tag2` `tag1 >= value1`
	functions	When executed, a function yields one value. Use parentheses to contain the operand of a function. Even though their syntax is similar, functions differ from instructions in that functions can only be used in expressions. Instructions cannot be used in expressions.	`function(tag1)`

Term:	Definition:	Examples:
instruction (see page 201)	An instruction is a standalone statement. An instruction uses parenthesis to contain its operands. Depending on the instruction, there can be zero, one, or multiple operands. When executed, an instruction yields one or more values that are part of a data structure. Terminate the instruction with a semi colon ";". Even though their syntax is similar, instructions differ from functions in that instructions cannot be used in expressions. Functions can only be used in expressions.	*instruction();* *instruction(operand);* *instruction(operand1, operand2,operand3);*
construct (see page 202)	A conditional statement used to trigger Structured Text code (i.e, other statements). Terminate the construct with a semi colon ";".	IF...THEN CASE FOR...DO WHILE...DO REPEAT...UNTIL EXIT
comment (see page 218)	Text that explains or clarifies what a section of Structured Text does. • Use comments to make it easier to interpret the Structured Text. • Comments do not affect the execution of the Structured Text. • Comments can appear anywhere in Structured Text.	*//comment* *(*start of comment . . . end of comment*)* */*start of comment . . . end of comment*/*

Assignments

Use an assignment to change the value stored within a tag. An assignment has this syntax:

tag := *expression* ;

where:

Component:	Description:	
tag	represents the tag that is getting the new value the tag must be a BOOL, SINT, INT, DINT, or REAL	
:=	is the assignment symbol	
expression	represents the new value to assign to the tag	
	If *tag* is this data type:	**Use this type of expression:**
	BOOL	BOOL expression
	SINT INT DINT REAL	numeric expression
;	ends the assignment	

The *tag* retains the assigned value until another assignment changes the value.

The expression can be simple, such as an immediate value or another tag name, or the expression can be complex and include several operators and/or functions. See the next section "Expressions"on page 6-194 for details.

TIP

I/O module data updates asynchronously to the execution of logic. If you reference an input multiple times in your logic, the input could change state between separate references. If you need the input to have the same state for each reference, buffer the input value and reference that buffer tag. For more information on buffering data, see page28.

Specify a non-retentive assignment

The non-retentive assignment is different from the regular assignment described above in that the tag in a non-retentive assignment is reset to zero each time the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for *Automatic reset* (This applies only if you embed the assignment in the action of the step or use the action to call a Structured Text routine via a JSR instruction.)

A non-retentive assignment has this syntax:

tag [:=] *expression* ;

where:

Component:	Description:	
tag	represents the tag that is getting the new value the tag must be a BOOL, SINT, INT, DINT, or REAL	
[:=]	is the non-retentive assignment symbol	
expression	represents the new value to assign to the tag	
	If *tag* is this data type:	**Use this type of expression:**
	BOOL	BOOL expression
	SINT INT DINT REAL	numeric expression
;	ends the assignment	

Assign an ASCII character to a string

Use the assignment operator to assign an ASCII character to an element of the DATA member of a string tag. To assign a character, specify the value of the character or specify the tag name, DATA member, and element of the character. For example:

This is OK:	This is *not* OK.
`string1.DATA[0]:= 65;`	`string1.DATA[0] := A;`
`string1.DATA[0]:= string2.DATA[0];`	`string1 := string2;`

To add or insert a string of characters to a string tag, use either of these ASCII string instructions:

To:	Use this instruction:
add characters to the end of a string	CONCAT
insert characters into a string	INSERT

Expressions

An expression is a tag name, equation, or comparison. To write an expression, use any of these elements:

- tag name that stores the value (variable)
- number that you enter directly into the expression (immediate value)
- functions, such as: ABS, TRUNC
- operators, such as: +, -, <, >, And, Or

As you write expressions, follow these general rules:

- Use any combination of upper-case and lower-case letter. For example, these three variations of "AND" are acceptable: AND, And, and.
- For more complex requirements, use parentheses to group expressions within expressions. This makes the whole expression easier to read and ensures that the expression executes in the desired sequence. See "Determine the order of execution"on 200.

In Structured Text, you use two types of expressions:

BOOL expression: An expression that produces either the BOOL value of 1 (true) or 0 (false).

- A bool expression uses bool tags, relational operators, and logical operators to compare values or check if conditions are true or false. For example, `tag1>65`.
- A simple bool expression can be a single BOOL tag.
- Typically, you use bool expressions to condition the execution of other logic.

Numeric expression: An expression that calculates an integer or floating-point value.

- A numeric expression uses arithmetic operators, arithmetic functions, and bitwise operators. For example, `tag1+5`.
- Often, you nest a numeric expression within a bool expression. For example, `(tag1+5)>65`.

Use the following table to choose operators for your expressions:

If you want to:	Then:
Calculate an arithmetic value	“Use arithmetic operators and functions“on page 6-196.
Compare two values or strings	“Use relational operators“on page 6-197.
Check if conditions are true or false	“Use logical operators“on page 6-199.
Compare the bits within values	“Use bitwise operators“on page 6-200.

Use arithmetic operators and functions

You can combine multiple operators and functions in arithmetic expressions.

Arithmetic operators calculate new values.

To:	Use this operator:	Optimal data type:
add	+	DINT, REAL
subtract/negate	-	DINT, REAL
multiply	*	DINT, REAL
exponent (x to the power of y)	**	DINT, REAL
divide	/	DINT, REAL
modulo-divide	MOD	DINT, REAL

Arithmetic functions perform math operations. Specify a constant, a non-boolean tag, or an expression for the function.

For:	Use this function:	Optimal data type:
absolute value	ABS (*numeric_expression*)	DINT, REAL
arc cosine	ACOS (*numeric_expression*)	REAL
arc sine	ASIN (*numeric_expression*)	REAL
arc tangent	ATAN (*numeric_expression*)	REAL
cosine	COS (*numeric_expression*)	REAL
radians to degrees	DEG (*numeric_expression)*	DINT, REAL
natural log	LN (*numeric_expression*)	REAL
log base 10	LOG (*numeric_expression*)	REAL
degrees to radians	RAD (*numeric_expression*)	DINT, REAL
sine	SIN (*numeric_expression*)	REAL
square root	SQRT (*numeric_expression*)	DINT, REAL
tangent	TAN (*numeric_expression*)	REAL
truncate	TRUNC (*numeric_expression*)	DINT, REAL

For example:

Use this format:	Example:	
	For this situation:	You'd write:
value1 operator value2	If gain_4 and gain_4_adj are DINT tags and your specification says: "Add 15 to gain_4 and store the result in gain_4_adj."	`gain_4_adj := gain_4+15;`
operator value1	If alarm and high_alarm are DINT tags and your specification says: "Negate high_alarm and store the result in alarm."	`alarm:= -high_alarm;`
function(numeric_expression)	If overtravel and overtravel_POS are DINT tags and your specification says: "Calculate the absolute value of overtravel and store the result in overtravel_POS."	`overtravel_POS := ABS(overtravel);`
value1 operator (function((value2+value3)/2)	If adjustment and position are DINT tags and sensor1 and sensor2 are REAL tags and your specification says: "Find the absolute value of the average of sensor1 and sensor2, add the adjustment, and store the result in position."	`position := adjustment + ABS((sensor1 + sensor2)/2);`

Use relational operators

Relational operators compare two values or strings to provide a true or false result. The result of a relational operation is a BOOL value:

If the comparison is:	The result is:
true	1
false	0

Use these relational operators:

For this comparison:	Use this operator:	Optimal Data Type:
equal	=	DINT, REAL, string
less than	<	DINT, REAL, string
less than or equal	<=	DINT, REAL, string
greater than	>	DINT, REAL, string
greater than or equal	>=	DINT, REAL, string
not equal	<>	DINT, REAL, string

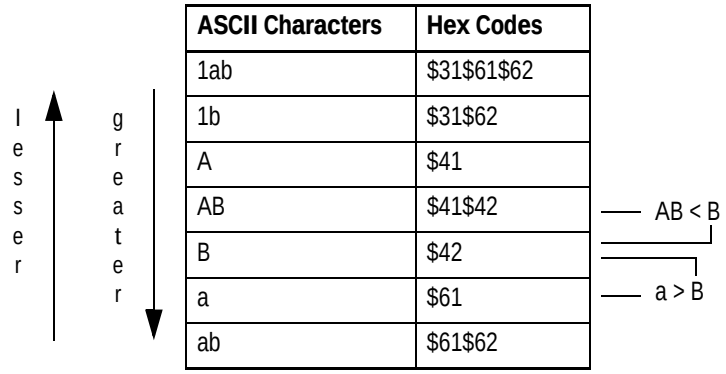
For example:

Use this format:	Example:	
	For this situation:	You'd write:
value1 operator value2	If temp is a DINT tag and your specification says: "If temp is less than 100° then…"	`IF temp<100 THEN...`
stringtag1 operator stringtag2	If bar_code and dest are string tags and your specification says: "If bar_code equals dest then…"	`IF bar_code=dest THEN...`
char1 operator char2 To enter an ASCII character directly into the expression, enter the decimal value of the character.	If bar_code is a string tag and your specification says: "If bar_code.DATA[0] equals 'A' then…"	`IF bar_code.DATA[0]=65 THEN...`
bool_tag := bool_expressions	If count and length are DINT tags, done is a BOOL tag, and your specification says "If count is greater than or equal to length, you are done counting."	`done := (count >= length);`

How Strings Are Evaluated

The hexadecimal values of the ASCII characters determine if one string is less than or greater than another string.

- When the two strings are sorted as in a telephone directory, the order of the strings determines which one is greater.



ASCII Characters	Hex Codes
1ab	$31$61$62
1b	$31$62
A	$41
AB	$41$42
B	$42
a	$61
ab	$61$62

- Strings are equal if their characters match.
- Characters are case sensitive. Upper case "A" ($41) is *not* equal to lower case "a" ($61).

Use logical operators

Logical operators let you check if multiple conditions are true or false. The result of a logical operation is a BOOL value:

If the comparison is:	The result is:
true	1
false	0

Use these logical operators:

For:	Use this operator:	Data Type:
logical AND	&, AND	BOOL
logical OR	OR	BOOL
logical exclusive OR	XOR	BOOL
logical complement	NOT	BOOL

For example:

Use this format:	Example:	
	For this situation:	You'd write:
BOOLtag	If photoeye is a BOOL tag and your specification says: "If photoeye_1 is on then…"	`IF photoeye THEN...`
NOT *BOOLtag*	If photoeye is a BOOL tag and your specification says: "If photoeye is off then…"	`IF NOT photoeye THEN...`
expression1 & *expression2*	If photoeye is a BOOL tag, temp is a DINT tag, and your specification says: "If photoeye is on and temp is less than 100° then…".	`IF photoeye & (temp<100) THEN...`
expression1 OR *expression2*	If photoeye is a BOOL tag, temp is a DINT tag, and your specification says: "If photoeye is on or temp is less than 100° then…".	`IF photoeye OR (temp<100) THEN...`
expression1 XOR *expression2*	If photoeye1 and photoeye2 are BOOL tags and your specification says: "If: • photoeye1 is on while photoeye2 is off or • photoeye1 is off while photoeye2 is on then…"	`IF photoeye1 XOR photoeye2 THEN...`
BOOLtag := *expression1* & *expression2*	If photoeye1 and photoeye2 are BOOL tags, open is a BOOL tag, and your specification says: "If photoeye1 and photoeye2 are both on, set open to true".	`open := photoeye1 & photoeye2;`

Use bitwise operators

Bitwise operators manipulate the bits within a value based on two values.

For:	Use this operator:	Optimal Data Type:
bitwise AND	&, AND	DINT
bitwise OR	OR	DINT
bitwise exclusive OR	XOR	DINT
bitwise complement	NOT	DINT

For example:

Use this format:	Example:	
	For this situation:	You'd write:
value1 operator value2	If input1, input2, and result1 are DINT tags and your specification says: "Calculate the bitwise result of input1 and input2. Store the result in result1."	`result1 := input1 AND input2;`

Determine the order of execution

The operations you write into an expression are performed in a prescribed order, not necessarily from left to right.

- Operations of equal order are performed from left to right.
- If an expression contains multiple operators or functions, group the conditions in parenthesis "( )". This ensures the correct order of execution and makes it easier to read the expression.

Order:	Operation:
1.	( )
2.	function (…)
3.	**
4.	– (negate)
5.	NOT
6.	*, /, MOD
7.	+, - (subtract)
8.	<, <=, >, >=
9.	=, <>
10.	&, AND
11.	XOR
12.	OR

Instructions

Structured Text statements can also be instructions. See the Locator Table at the beginning of this manual for a list of the instructions available in Structured Text. A Structured Text instruction executes each time it is scanned. A Structured Text instruction within a construct executes every time the conditions of the construct are true. If the conditions of the construct are false, the statements within the construct are not scanned. There is no rung-condition or state transition that triggers execution.

This differs from Function Block instructions that use EnableIn to trigger execution. Structured Text instructions execute as if EnableIn is always set.

This also differs from relay ladder instructions that use rung-condition-in to trigger execution. Some relay ladder instructions only execute when rung-condition-in toggles from false to true. These are transitional relay ladder instructions. In Structured Text, instructions will execute each time they are scanned unless you pre-condition the execution of the Structured Text instruction.

For example, the ABL instruction is a transitional instruction in relay ladder. In this example, the ABL instruction only executes on a scan when tag_xic transitions from cleared to set. The ABL instruction does not execute when tag_xic stays set or when tag_xic is cleared.



In Structured Text, if you write this example as:

```
IF tag_xic THEN ABL(0,serial_control);

END_IF;
```

the ABL instruction will execute every scan that tag_xic is set, not just when tag_xic transitions from cleared to set.

If you want the ABL instruction to execute only when tag_xic transitions from cleared to set, you have to condition the Structured Text instruction. Use a one shot to trigger execution.

```
osri_1.InputBit := tag_xic;
OSRI(osri_1);

IF (osri_1.OutputBit) THEN
     ABL(0,serial_control);
END_IF;
```

Constructs

Constructs can be programmed singly or nested within other constructs.

If you want to:	Use this construct:	Available in these languages:	Page
do something if or when specific conditions occur	IF...THEN	Structured Text	6-203
select what to do based on a numerical value	CASE...OF	Structured Text	6-206
do something a specific number of times before doing anything else	FOR...DO	Structured Text	6-209
keep doing something as long as certain conditions are true	WHILE...DO	Structured Text	6-212
keep doing something until a condition is true	REPEAT...UNTIL	Structured Text	6-215

Some key words are reserved for future use

These constructs are not available:

- GOTO
- REPEAT

RSLogix 5000 software will not let you use them.

IF...THEN

Use IF…THEN to do something if or when specific conditions occur.

Operands:

Structured Text

```
IF bool_expression THEN
    <statement>;
END_IF;
```

Operand:	Type:	Format:	Enter:
bool_ expression	BOOL	tag expression	BOOL tag or expression that evaluates to a BOOL value (BOOL expression)

Description:

The syntax is:

```
IF bool_expression1 THEN
      <statement >;          <-- statements to execute when bool_expression1 is true
                .
                .
                .
optional { ELSIF bool_expression2 THEN
      <statement>;           <-- statements to execute when bool_expression2 is true
                .
                .
                .
optional { ELSE
      <statement>;           <-- statements to execute when both expressions are false
                .
                .
                .
END_IF;
```

To use ELSIF or ELSE, follow these guidelines:

1. To select from several possible groups of statements, add one or more ELSIF statements.
   - Each ELSIF represents an alternative path.
   - Specify as many ELSIF paths as you need.
   - The controller executes the first true IF or ELSIF and skips the rest of the ELSIFs and the ELSE.
2. To do something when all of the IF or ELSIF conditions are false, add an ELSE statement.

This table summarizes combinations of IF, THEN, ELSIF, and ELSE.

If you want to:	And:	Then use this construct
do something if or when conditions are true	do nothing if conditions are false	IF…THEN
	do something else if conditions are false	IF…THEN…ESLE
choose from alternative statements (or groups of statements) based on input conditions	do nothing if conditions are false	IF…THEN…ELSIF
	assign default statements if all conditions are false	IF…THEN…ELSIF…ELSE

Arithmetic Status Flags: not affected

Fault Conditions: none

Example 1: IF…THEN

If you want this:	Enter this Structured Text:
IF rejects > 3 then conveyor = off (0) alarm = on (1)	`IF rejects > 3 THEN` `  conveyor := 0;` `  alarm := 1;` `END_IF;`

Example 2: IF…THEN…ELSE

If you want this:	Enter this Structured Text:
If conveyor direction contact = forward (1) then light = off Otherwise light = on	`IF conveyor_direction THEN` `  light := 0;` `ELSE` `  light [:=] 1;` `END_IF;`

The [:=] tells the controller to clear light whenever the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for Automatic reset (This applies only if you embed the assignment in the action of the step or use the action to call a Structured Text routine via a JSR instruction.)

Example 3: IF…THEN…ELSIF

If you want this:	Enter this Structured Text:
If sugar low limit switch = low (on) and sugar high limit switch = not high (on) then	`IF Sugar.Low & Sugar.High THEN`
inlet valve = open (on)	`Sugar.Inlet [:=] 1;`
Until sugar high limit switch = high (off)	`ELSIF NOT(Sugar.High) THEN`
	`Sugar.Inlet := 0;`
	`END_IF;`

The [:=] tells the controller to clear Sugar.Inlet whenever the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for Automatic reset (This applies only if you embed the assignment in the action of the step or use the action to call a Structured Text routine via a JSR instruction.)

Example 4: IF…THEN…ELSIF…ELSE

If you want this:	Enter this Structured Text:
If tank temperature > 100	`IF tank.temp > 200 THEN`
then pump = slow	`pump.fast :=1; pump.slow :=0; pump.off :=0;`
If tank temperature > 200	`ELSIF tank.temp > 100 THEN`
then pump = fast	`pump.fast :=0; pump.slow :=1; pump.off :=0;`
otherwise pump = off	`ELSE`
	`pump.fast :=0; pump.slow :=0; pump.off :=1;`
	`END_IF;`

CASE...OF

Use CASE to select what to do based on a numerical value.

Operands:



```
CASE numeric_expression OF
     selector1: statement;
     selectorN: statement;
ELSE
     statement;
END_CASE;
```

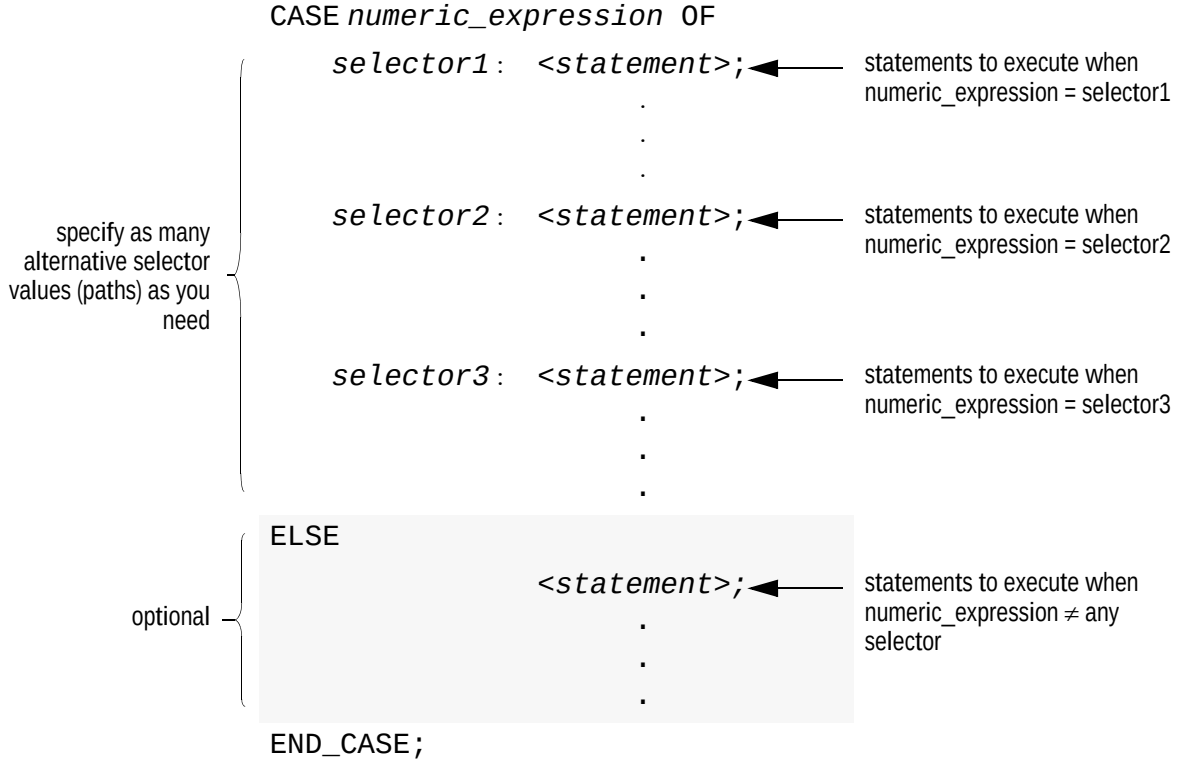
Structured Text

Operand:	Type:	Format:	Enter:
numeric_expression	SINT INT DINT REAL	tag expression	tag or expression that evaluates to a number (numeric expression)
selector	SINT INT DINT REAL	immediate	same type as *numeric_expression*

IMPORTANT If you use REAL values, use a range of values for a selector because a REAL value is more likely to be within a range of values than an exact match of one, specific value.

Description:

The syntax is:



See the table on the next page for valid selector values.

The syntax for entering the selector values is:

When selector is:	Enter:
one value	value: statement
multiple, distinct values	value1, value2, valueN : <statement> Use a comma (,) to separate each value.
a range of values	value1...valueN : <statement> Use two periods (..) to identify the range.
distinct values plus a range of values	valuea, valueb, value1...valueN : <statement>

The CASE construct is similar to a switch statement in the C or C++ programming languages. However, with the CASE construct the controller executes only the statements that are associated with the first matching selector value. Execution always breaks after the statements of that selector and goes to the END_CASE statement.

Arithmetic Status Flags: not affected

Fault Conditions: none

Example

If you want this:	Enter this Structured Text:
If recipe number = 1 then	`CASE recipe_number OF`
Ingredient A outlet 1 = open (1)	`1: Ingredient_A.Outlet_1 :=1;`
Ingredient B outlet 4 = open (1)	`Ingredient_B.Outlet_4 :=1;`
If recipe number = 2 or 3 then	`2,3: Ingredient_A.Outlet_4 :=1;`
Ingredient A outlet 4 = open (1)	`Ingredient_B.Outlet_2 :=1;`
Ingredient B outlet 2 = open (1)	
If recipe number = 4, 5, 6, or 7 then	`4..7: Ingredient_A.Outlet_4 :=1;`
Ingredient A outlet 4 = open (1)	`Ingredient_B.Outlet_2 :=1;`
Ingredient B outlet 2 = open (1)	
If recipe number = 8, 11, 12, or 13 then	`8,11..13 Ingredient_A.Outlet_1 :=1;`
Ingredient A outlet 1 = open (1)	`Ingredient_B.Outlet_4 :=1;`
Ingredient B outlet 4 = open (1)	
Otherwise all outlets = closed (0)	`ELSE`
	`Ingredient_A.Outlet_1 [:=]0;`
	`Ingredient_A.Outlet_4 [:=]0;`
	`Ingredient_B.Outlet_2 [:=]0;`
	`Ingredient_B.Outlet_4 [:=]0;`
	`END_CASE;`

The [:=] tells the controller to also clear the outlet tags whenever the controller:

- enters the RUN mode
- leaves the step of an SFC if you configure the SFC for Automatic reset (This applies only if you embed the assignment in the action of the step or use the action to call a Structured Text routine via a JSR instruction.)

FOR…DO

Use the FOR…DO loop to do something a specific number of times before doing anything else.

Operands:



```
FOR count:= initial_value TO
final_value BY increment DO
    <statement>;
END_FOR;
```

Structured Text

Operand:	Type:	Format:	Description:
count	SINT INT DINT	tag	tag to store count position as the FOR…DO executes
initial_value	SINT INT DINT	tag expression immediate	must evaluate to a number specifies initial value for count
final_value	SINT INT DINT	tag expression immediate	specifies final value for count, which determines when to exit the loop
increment	SINT INT DINT	tag expression immediate	(*optional*) amount to increment count each time through the loop If you don't specify an increment, the count increments by 1.

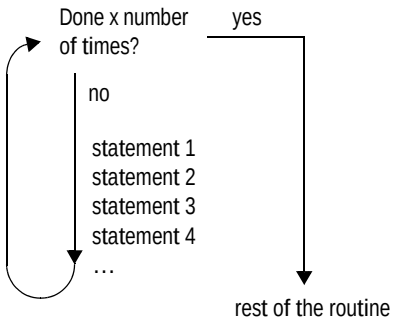
IMPORTANT Make sure that you do not iterate within the loop too many times in a single scan.

- The controller does not execute any other statements in the routine until it completes the loop.
- If the time that it takes to complete the loop is greater than the watchdog timer for the task, a major fault occurs.
- Consider using a different construct, such as IF...THEN.

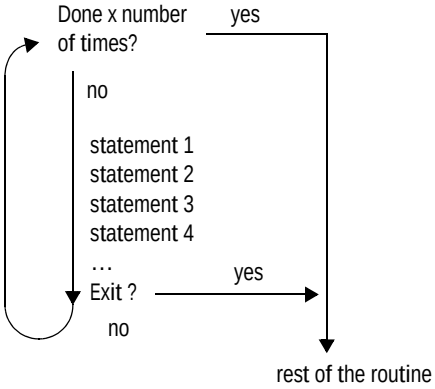
Description: The syntax is:

```
FOR count := initial_value
    TO final_value
optional { BY increment                    If you don't specify an increment, the loop
                                           increments by 1.
    DO
    <statement>;
optional { IF bool_expression THEN
             EXIT;  <-------------- If there are conditions when you want to
                                    exit the loop early, use other statements,
    END_IF;                         such as an IF...THEN construct, to
                                    condition an EXIT statement.
END_FOR;
```

These diagrams show how a FOR...DO loop executes and how an EXIT statement leaves the loop early.



The FOR…DO loop executes a specific number of times.



To stop the loop before the count reaches the last value, use an EXIT statement.

Arithmetic Status Flags:

not affected

Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
the construct loops too long	6	1

Example 1:

If you want this:	Enter this Structured Text:
Clear bits 0 - 31 in an array of BOOLs: 1. Initialize the subscript tag to 0. 2. Clear array[subscript] . For example, when subscript = 5, clear array[5]. 3. Add 1 to subscript. 4. If subscript is £ to 31, repeat 2 and 3. Otherwise, stop.	`For subscript:=0 to 31 by 1 do` `  array[subscript] := 0;` `End_for;`

Example 2:

If you want this:	Enter this Structured Text:
A user-defined data type (structure) stores this information about an item in your inventory: • Barcode ID of the item (string data type) • Quantity in stock of the item (DINT data type) An array of the above structure contains an element for each different item in your inventory. You want to search the array for a specific product (use its bar code) and determine the quantity that is in stock. 1. Get the size (number of items) of the Inventory array and store the result in Inventory_Items (DINT tag). 2. Initialize the position tag to 0. 3. If Barcode matches the ID of an item in the array, then: a. Set the Quantity tag = Inventory[position].Qty. This produces the quantity in stock of the item. b. Stop. Barcode is a string tag that stores the bar code of the item for which you are searching. For example, when position = 5, compare Barcode to Inventory[5].ID. 4. Add 1 to position. 5. If position is £ to (Inventory_Items -1), repeat 3 and 4. Since element numbers start at 0, the last element is 1 less than the number of elements in the array. Otherwise, stop.	`SIZE(Inventory,0,Inventory_Items);` `For position:=0 to Inventory_Items - 1 do` `  If Barcode = Inventory[position].ID then` `  Quantity := Inventory[position].Qty;` `  Exit;` `  End_if;` `End_for;`

WHILE...DO

Use the WHILE…DO loop to keep doing something as long as certain conditions are true.

Operands:

Structured Text

```
WHILE bool_expression DO
    <statement>;
END_WHILE;
```

Operand:	Type:	Format:	Enter:
bool_ expression	BOOL	tag expression	BOOL tag or expression that evaluates to a BOOL value

IMPORTANT Make sure that you do not iterate within the loop too many times in a single scan.

- The controller does not execute any other statements in the routine until it completes the loop.
- If the time that it takes to complete the loop is greater than the watchdog timer for the task, a major fault occurs.
- Consider using a different construct, such as IF...THEN.

Description: The syntax is:

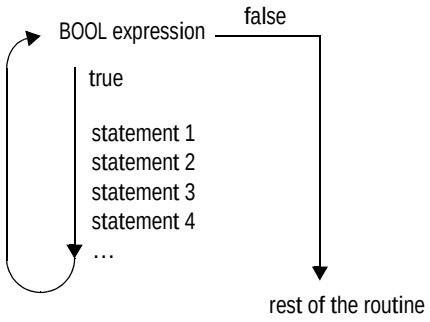
```
WHILE bool_expression1 DO
   <statement>;
   IF bool_expression2 THEN
     EXIT;
   END_IF;
END_WHILE;
```

statements to execute while bool_expression1 is true

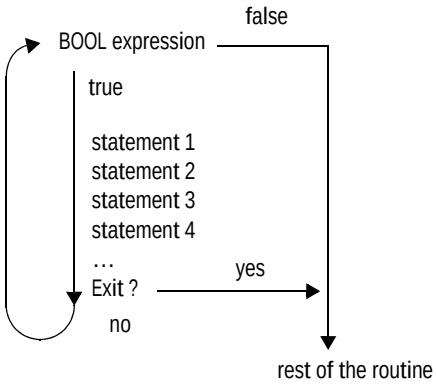
optional

If there are conditions when you want to exit the loop early, use other statements, such as an IF...THEN construct, to condition an EXIT statement.

These diagrams show how a WHILE...DO loop executes and how an EXIT statement leaves the loop early.



While the *bool_expression* is true, the controller executes only the statements within the WHILE…DO loop.



To stop the loop before the conditions are true, use an EXIT statement.

Arithmetic Status Flags: not affected

Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
the construct loops too long	6	1

Example 1:

If you want this:	Enter this Structured Text:
The WHILE...DO loop evaluates its conditions first. If the conditions are true, the controller then executes the statements within the loop. This differs from the REPEAT...UNTIL loop because the REPEAT...UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again. The statements in a REPEAT...UNTIL loop are always executed at least once. The statements in a WHILE...DO loop might never be executed.	`pos := 0;` `While ((pos <= 100) & structarray[pos].value <> targetvalue)) do` `  pos := pos + 2;` `  String_tag.DATA[pos] := SINT_array[pos];` `end_while;`

Example 2:

If you want this:	Enter this Structured Text:
Move ASCII characters from a SINT array into a string tag. (In a SINT array, each element holds one character.) Stop when you reach the carriage return. 1. Initialize Element_number to 0. 2. Count the number of elements in SINT_array (array that contains the ASCII characters) and store the result in SINT_array_size (DINT tag). 3. If the character at SINT_array[element_number] = 13 (decimal value of the carriage return), then stop. 4. Set String_tag[element_number] = the character at SINT_array[element_number]. 5. Add 1 to element_number. This lets the controller check the next character in SINT_array. 6. Set the Length member of String_tag = element_number. (This records the number of characters in String_tag so far.) 7. If element_number = SINT_array_size, then stop. (You are at the end of the array and it does not contain a carriage return.) 8. Go to 3.	`element_number := 0;` `SIZE(SINT_array, 0, SINT_array_size);` `While SINT_array[element_number] <> 13 do` `  String_tag.DATA[element_number] := SINT_array[element_number];` `  element_number := element_number + 1;` `  String_tag.LEN := element_number;` `  If element_number = SINT_array_size then` `  exit;` `  end_if;` `end_while;`

REPEAT...UNTIL

Use the REPEAT…UNTIL loop to keep doing something until conditions are true.

Operands:

Structured Text

```
REPEAT
    <statement>;
UNTIL bool_expression
END_REPEAT;
```

Operand:	Type:	Format:	Enter:
bool_ expression	BOOL	tag expression	BOOL tag or expression that evaluates to a BOOL value (BOOL expression)

IMPORTANT Make sure that you do not iterate within the loop too many times in a single scan.

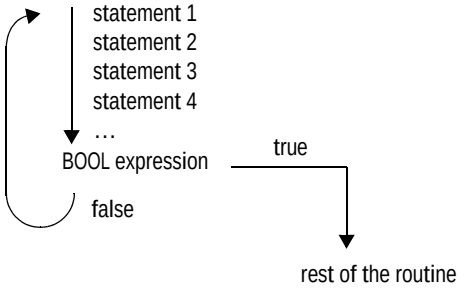
- The controller does not execute any other statements in the routine until it completes the loop.
- If the time that it takes to complete the loop is greater than the watchdog timer for the task, a major fault occurs.
- Consider using a different construct, such as IF...THEN.

Description: The syntax is:

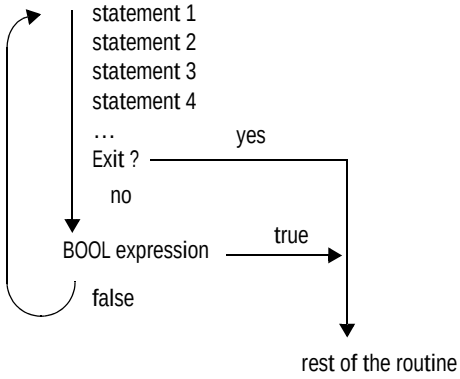
```
REPEAT
   <statement>;          <-- statements to execute while bool_expression1 is false
   IF bool_expression2 THEN
     EXIT;               <-- If there are conditions when you want to exit the loop early, use other statements, such as an IF...THEN construct, to condition an EXIT statement.
   END_IF;
UNTIL bool_expression1
END_REPEAT;
```

(IF...END_IF: optional)

These diagrams show how a REPEAT...UNTIL loop executes and how an EXIT statement leaves the loop early.



While the *bool_expression* is false, the controller executes only the statements within the REPEAT…UNTIL loop.



To stop the loop before the conditions are false, use an EXIT statement.

Arithmetic Status Flags: not affected

Fault Conditions:

A major fault will occur if:	Fault type:	Fault code:
the construct loops too long	6	1

Example 1:

If you want this:	Enter this Structured Text:
The REPEAT...UNTIL loop executes the statements in the construct and then determines if the conditions are true before executing the statements again. This differs from the WHILE...DO loop because the WHILE...DO The WHILE...DO loop evaluates its conditions first. If the conditions are true, the controller then executes the statements within the loop. The statements in a REPEAT...UNTIL loop are always executed at least once. The statements in a WHILE...DO loop might never be executed.	`pos := -1;` `REPEAT` `  pos := pos + 2;` `UNTIL ((pos = 101) OR (structarray[pos].value = targetvalue))` `end_repeat;`

Example 2:

If you want this:	Enter this Structured Text:
Move ASCII characters from a SINT array into a string tag. (In a SINT array, each element holds one character.) Stop when you reach the carriage return. 1. Initialize Element_number to 0. 2. Count the number of elements in SINT_array (array that contains the ASCII characters) and store the result in SINT_array_size (DINT tag). 3. Set String_tag[element_number] = the character at SINT_array[element_number]. 4. Add 1 to element_number. This lets the controller check the next character in SINT_array. 5. Set the Length member of String_tag = element_number. (This records the number of characters in String_tag so far.) 6. If element_number = SINT_array_size, then stop. (You are at the end of the array and it does not contain a carriage return.) 7. If the character at SINT_array[element_number] = 13 (decimal value of the carriage return), then stop. Otherwise, go to 3.	`element_number := 0;` `SIZE(SINT_array, 0, SINT_array_size);` `Repeat` `  String_tag.DATA[element_number] := SINT_array[element_number];` `  element_number := element_number + 1;` `  String_tag.LEN := element_number;` `  If element_number = SINT_array_size then` `  exit;` `  end_if;` `Until SINT_array[element_number] = 13` `end_repeat;`

Comments

To make your Structured Text easier to interpret, add comments to it.

- Comments let you use plain language to describe how your Structured Text works.
- Comments do not affect the execution of the Structured Text.

Structured Text comments are downloaded into controller memory and are available for upload. To add comments to your Structured Text:

To add a comment:	Use one of these formats:
on a single line	`//comment`
at the end of a line of Structured Text	`(*comment*)` `/*comment*/`
within a line of Structured Text	`(*comment*)` `/*comment*/`
that spans more than one line	`(*start of comment . . . end of comment*)` `/*start of comment . . . end of comment*/`

For example:

Format:	Example:
`//comment`	**At the beginning of a line** `//Check conveyor belt direction` `IF conveyor_direction THEN...` **At the end of a line** `ELSE //If conveyor isn't moving, set alarm light` `light := 1;` `END_IF;`
`(*comment*)`	`Sugar.Inlet[:=]1;(*open the inlet*)` `IF Sugar.Low (*low level LS*)& Sugar.High (*high level LS*)THEN...` `(*Controls the speed of the recirculation pump. The speed depends on the temperature in the tank.*)` `IF tank.temp > 200 THEN...`
`/*comment*/`	`Sugar.Inlet:=0;/*close the inlet*/` `IF bar_code=65 /*A*/ THEN...` `/*Gets the number of elements in the Inventory array and stores the value in the Inventory_Items tag*/` `SIZE(Inventory,0,Inventory_Items);`

Programming Ladder Diagram

When to Use This Chapter

Use this chapter to:

- develop the logic for a ladder diagram routine
- enter the logic into the routine

To program a ladder diagram routine:

Topic	Page
Definitions	219
Write Ladder Logic	222
Enter Ladder Logic	225
Assign Instruction Operands	227
Verify the Routine	7-229

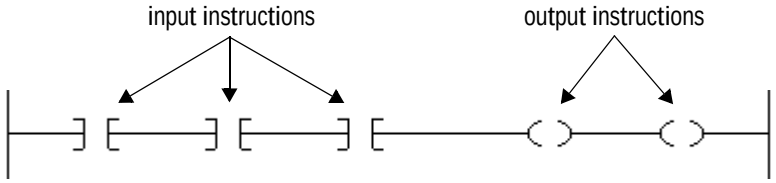
Definitions

Instruction

You organize ladder diagram as rungs on a ladder and put instructions on each rung. There are two basic types of instructions:

Input instruction: An instruction that checks, compares, or examines specific conditions in your machine or process.

Output instruction: An instruction that takes some action, such as turn on a device, turn off a device, copy data, or calculate a value.

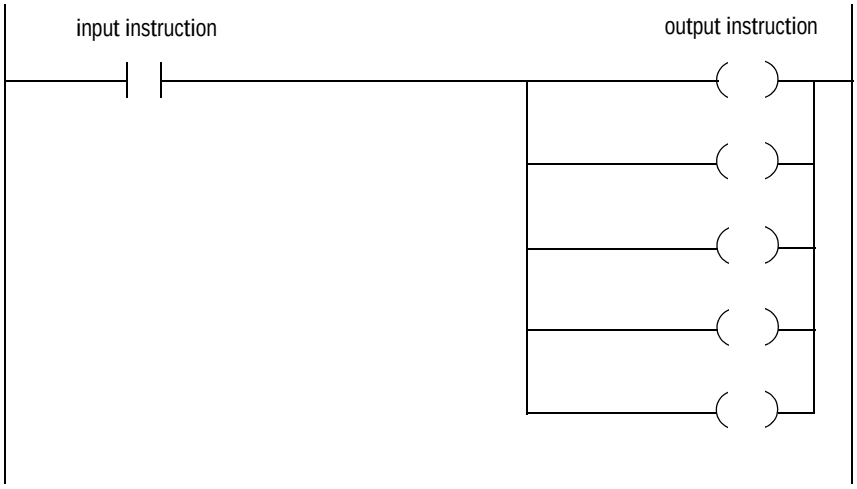


Branch

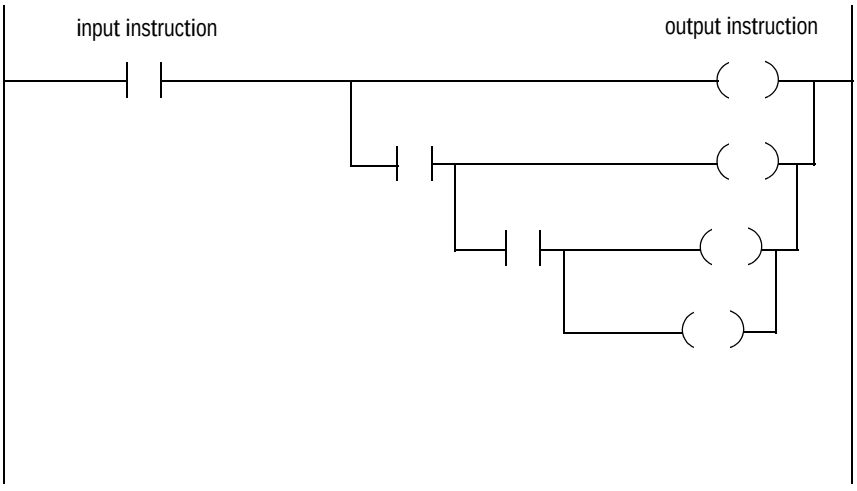
A branch is two or more instructions in parallel.



There is no limit to the number of parallel branch levels that you can enter. This example shows a parallel branch with five levels. The main rung is the first branch level, followed by four additional branches.



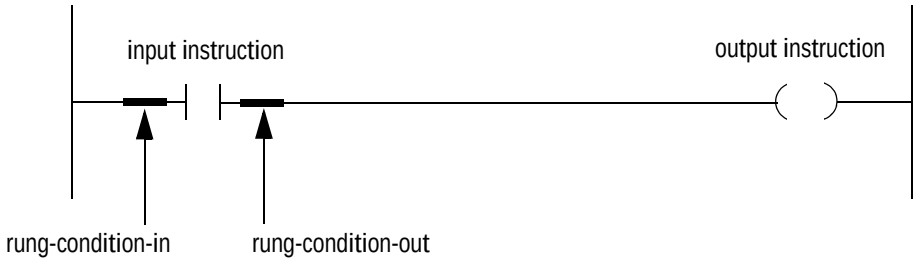
You can nest branches to as many as 6 levels. This example shows a nested branch. The bottom output instruction is on a nested branch that is three levels deep.



Large rungs with complex, nested branches result in having to scroll through the ladder editor and may end up spanning multiple pages when you print the logic. For easier maintainability, divide the logic into multiple smaller rungs.

Rung Condition

The controller evaluates ladder instructions based on the rung condition preceding the instruction (rung-condition-in).



Only input instructions affect the rung-condition-in of subsequent instructions on the rung:

- If the rung-condition-in to an input instruction is true, the controller evaluates the instruction and sets the rung-condition-out to match the results of the evaluation.
  - If the instruction evaluates to true, the rung-condition-out is true.
  - If the instruction evaluates to false, the rung-condition-out is false.
- An output instruction does not change the rung-condition-out.
  - If the rung-condition-in to an output instruction is true, the rung-condition-out is set to true.
  - If the rung-condition-in to an output instruction is false, the rung-condition-out is set to false.

Write Ladder Logic

Choose the Required Instructions

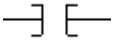
1. Separate the conditions to check from the action to take.
2. Choose the appropriate input instruction for each condition and the appropriate output instruction for each action.

   To choose specific instructions, see:

   - *Logix5000 Controllers General Instructions Reference Manual*, publication 1756-RM003
   - *Logix5000 Controllers Process and Drives Instructions Reference Manual*, publication 1756-RM006
   - *Logix5000 Controllers Motion Instruction Set Reference Manual*, publication 1756-RM007

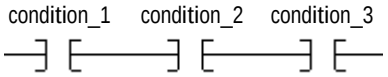
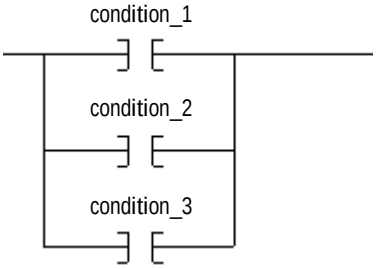
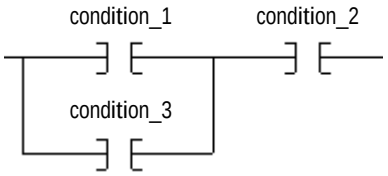
TIP I/O module data updates asynchronously to the execution of logic. If you reference an input multiple times in your logic, the input could change state between separate references. If you need the input to have the same state for each reference, buffer the input value and reference that buffer tag. For more information on buffering data, see page1-28.

The examples in this chapter use two simple instructions to help you learn how to write ladder diagram logic. The rules that you learn for these instructions apply to all other instructions.

Symbol:	Name:	Mnemonic:	Description:	
	Examine If Closed	XIC	An input instruction that looks at one bit of data.	
			If the bit is:	**Then the instruction (rung-condition-out) is:**
			on (1)	true
			off (0)	false
	Output Energize	OTE	An output instruction that controls one bit of data.	
			If the instructions to the left (rung-condition-in) are:	**Then the instruction turns the bit:**
			true	on (1)
			false	off (0)

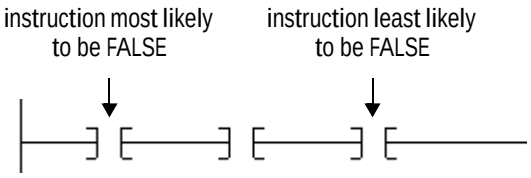
Arrange the Input Instructions

Determine how to arranged the input instructions on the rung:

To check multiple input conditions when:	Arrange the input instructions:
all conditions must be met in order to take action For example, If condition_1 AND condition_2 AND condition_3…	In series: 
any one of several conditions must be met in order to take action For example, If condition_1 OR condition_2 OR condition_3…	In parallel: 
there is a combination of the above For example, If condition_1 AND condition_2… OR If condition_3 AND condition_2…	In combination: 

TIP

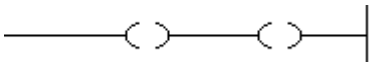
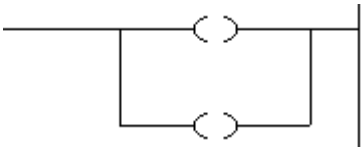
The controller executes all instructions on a rung regardless of their rung-condition-in. For optimal performance of a series of instructions, sequence the instructions from most likely to be false on the left to least likely to be false on the right.



When the controller finds a false instruction, it executes the remaining instructions in the series with their rung-condition-in set to false. Typically, an instruction executes faster when its rung-condition-in (rung) is false rather than true.

Arrange the Output Instructions

Place at least one output instruction to the right of the input instructions. You can enter multiple output instructions per rung of logic:

Option:	Example:
Place the output instructions in sequence on the rung (serial).	
Place the output instructions in branches (parallel).	
Place the output instructions between input instructions, as long as the last instruction on the rung is an output instruction.	

Choose a Tag Name for an Operand

Tag names follow these formats:

For a:	Specify:
tag	*tag_name*
bit number of a larger data type	*tag_name.bit_number*
member of a structure	*tag_name.member_name*
element of a one dimension array	*tag_name[x]*
element of a two dimension array	*tag_name[x,y]*
element of a three dimension array	*tag_name[x,y,z]*
element of an array within a structure	*tag_name.member_name[x]*
member of an element of an array	*tag_name[x,y,z].member_name*

where:

- *x* is the location of the element in the first dimension.
- *y* is the location of the element in the second dimension.
- *z* is the location of the element in the third dimension.

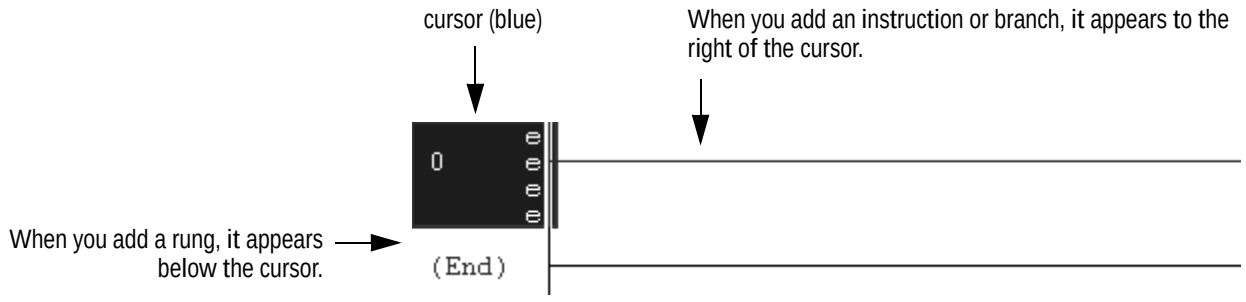
For a structure within a structure, add an additional *.member_name*.

EXAMPLE Choose a Tag Name for an Operand

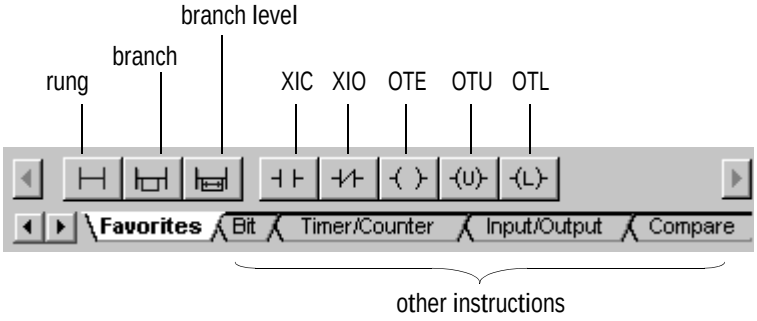
To access:	The tag name looks like this:
machine_on tag	machine_on
bit number 1 of the one_shots tag	one_shots.1
DN member (bit) of the running_seconds timer	running_seconds.DN
mix member of the north_tank tag	north_tank.mix
element 2 in the recipe array and element 1,1 in the tanks array	COP Copy File Source recipe[2] Dest tanks[1,1] Length 1
element 2 in the preset array within the north_tank tag	CLR Clear Dest north_tank.preset[2] 0
part_advance member of element 1 in the drill array	drill[1].part_advance

Enter Ladder Logic

A new routine contains a rung that is ready for instructions.



Use the Language Element toolbar to add a ladder diagram element.



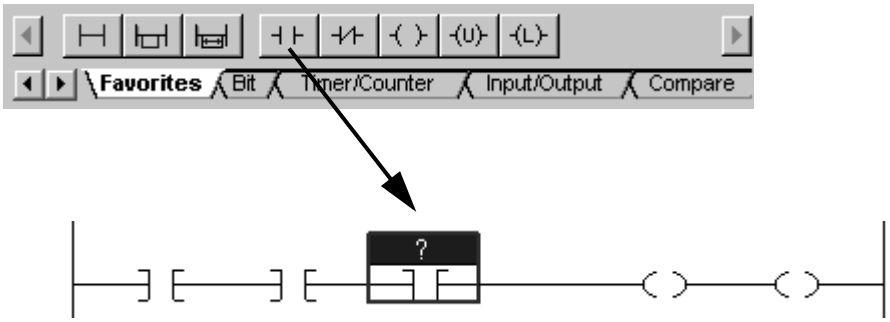
Append an Element to the Cursor Location

1. Click (select) the instruction, branch, or rung that is above or to the left of where you want to add an element.

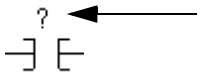
2. On the Language Element toolbar, click the button for the element that you want to add.

Drag and Drop an Element

Drag the button for the element directly to the desired location. A green dot shows a valid placement location (drop point). For example

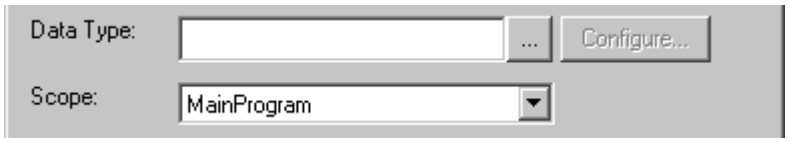


Assign Instruction Operands

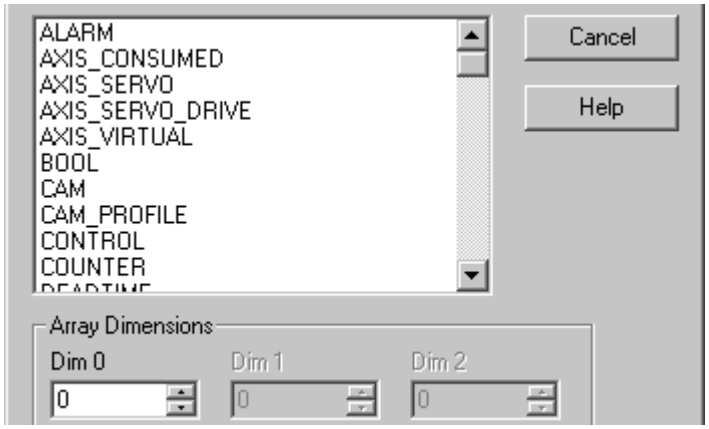


Create and Assign a New Tag

1. Click the operand area of the instruction.
2. Type a name for the tag and press [Enter].
3. Right-click the tag name and select New "tag_name".
4. Click the ... button.

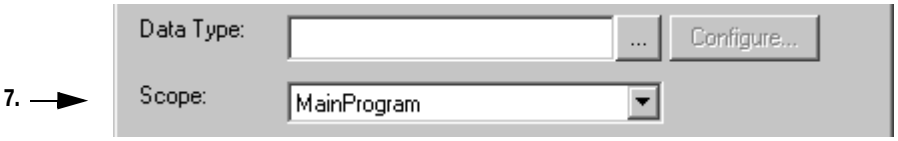


5. Select the data type for the tag.



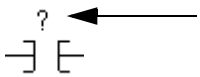
If you want to define the tag as an array, type the number of elements in each dimension.

6. Click OK
7. Choose the scope for the tag.



8. Click OK

Choose a Name or an Existing Tag



1. Double-click the operand area.

   A text entry box opens.

2. Click the t

3. Select the name:

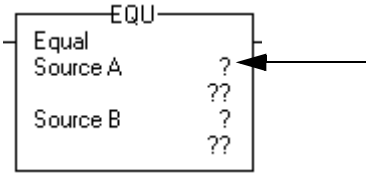
To select a:	Do this:
label, routine name, or similar type of name	Click the name.
tag	Double-click the tag name.
bit number	A. Click the tag name. B. To the right of the tag name, click t C. Click the required bit.

4. Press [Enter] or click a different spot on the diagram.

Drag a Tag From the Tags Window

1. Find the tag in the Tags window.
2. Click the tag two or three times until it highlights.
3. Drag the tag to its location on the instruction.

Assign an Immediate (Constant) Value



1. Click the operand area of the instruction.
2. Type the value and press [Enter].

Verify the Routine

As you program your routine (s), periodically verify your work:

1. In the top-most toolbar of the RSLogix 5000 window, click 
2. If any errors are listed at the bottom of the window:
   a. To go to the first error or warning, press [F4].
   b. Correct the error according to the description in the Results window.
   c. Go to step 1.
3. To close the Results window, press [Alt] + [1].

Notes:

Programming a Function Block Diagram

When to Use This Chapter

Use this chapter to:

- organize a Function Block routine
- develop one or more Function Block diagrams for the routine
- enter the Function Block diagrams into the routine

To program a Function Block diagram routine:

Topic	Page
Identify the Sheets for the Routine	231
Choose the Function Block Elements	232
Choose a Tag Name for an Element	233
Define the Order of Execution	235
Identify any Connectors	241
Define Program/Operator Control	242
Add a Sheet	245
Add a Function Block Element	245
Connect Elements	246
Assign a Tag	248
Assign an Immediate Value (Constant)	249
Connect Blocks with an OCON and ICON	250
Verify the Routine	251

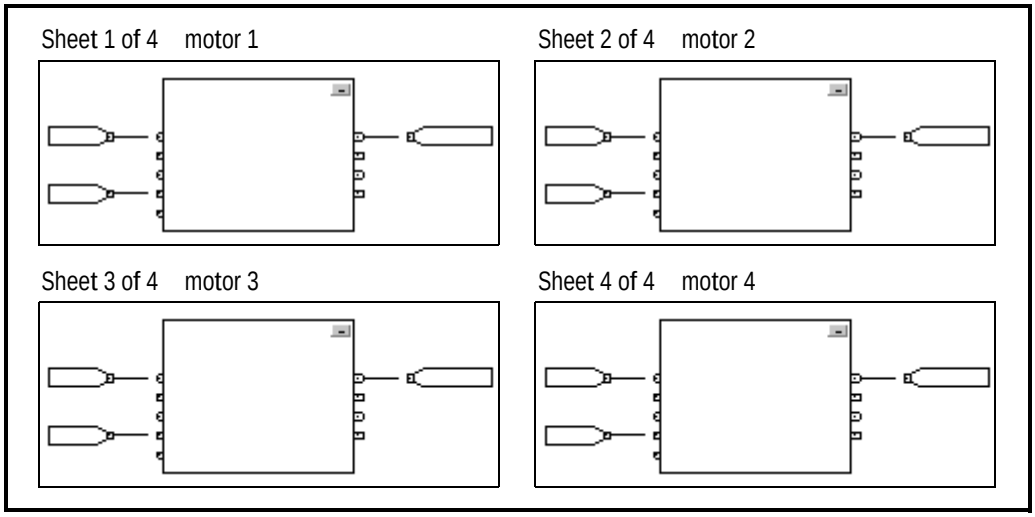
Identify the Sheets for the Routine

To make it easier to navigate through a Function Block routine, divide the routine into a series of sheets:

- Sheets help you organize and find your Function Blocks. They do not effect the order in which the Function Blocks execute.
- When the routine executes, all the sheets execute.
- In general, use one sheet for each device (motor, valve, etc.)

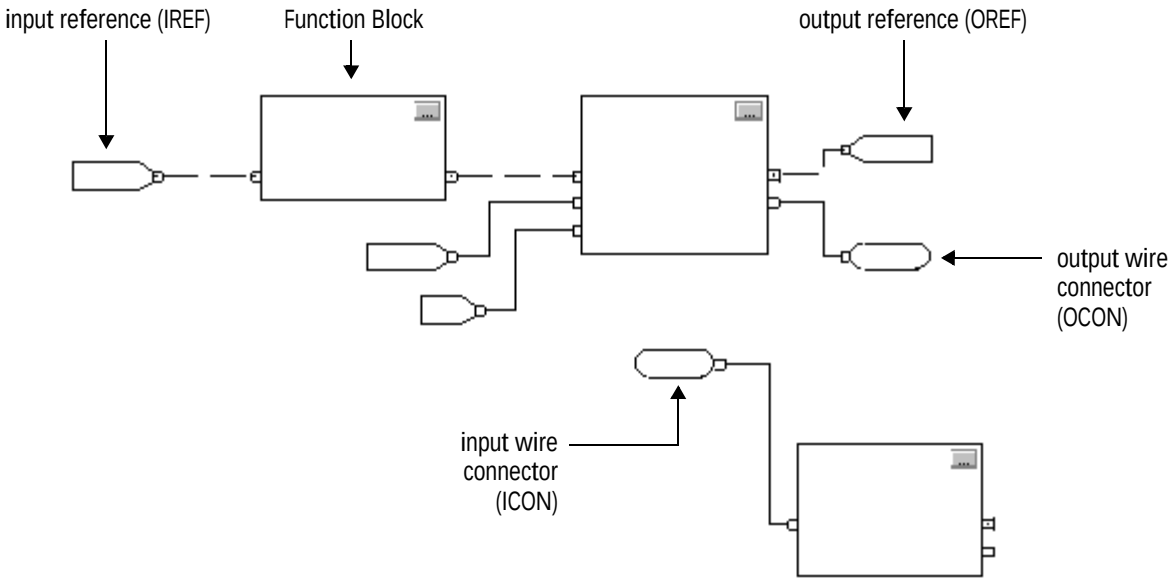
EXAMPLE Identify the Sheets for the Routine

Motor Control Routine



Choose the Function Block Elements

To control a device, use these elements:



To choose Function Block elements:

If you want to:	Then use a:
supply a value from an input device or tag	input reference (IREF)
send a value to an output device or tag	output reference (OREF)
perform an operation on an input value or values and produce an output value or values	Function Block
transfer data between Function Blocks when they are: • far apart on the same sheet • on different sheets within the same routine	output wire connector (OCON) and an input wire connector (ICON)
disperse data to several points in the routine	single output wire connector (OCON) and multiple input wire connectors (ICON)

Choose a Tag Name for an Element

Each Function Block uses a tag to store configuration and status information about the instruction.

- When you add Function Block instruction, RSLogix 5000 software automatically creates a tag for the block. You can use this tag as is, rename the tag, or assign a different tag.
- For IREFs and OREFs, you have to create a tag or assign an existing tag.

For a:	Specify:
tag	*tag_name*
bit number of a larger data type	*tag_name.bit_number*
member of a structure	*tag_name.member_name*
element of a one dimension array	*tag_name[x]*
element of a two dimension array	*tag_name[x,y]*
element of a three dimension array	*tag_name[x,y,z]*
element of an array within a structure	*tag_name.member_name[x]*
member of an element of an array	*tag_name[x,y,z].member_name*

where:

x is the location of the element in the first dimension.

y is the location of the element in the second dimension.

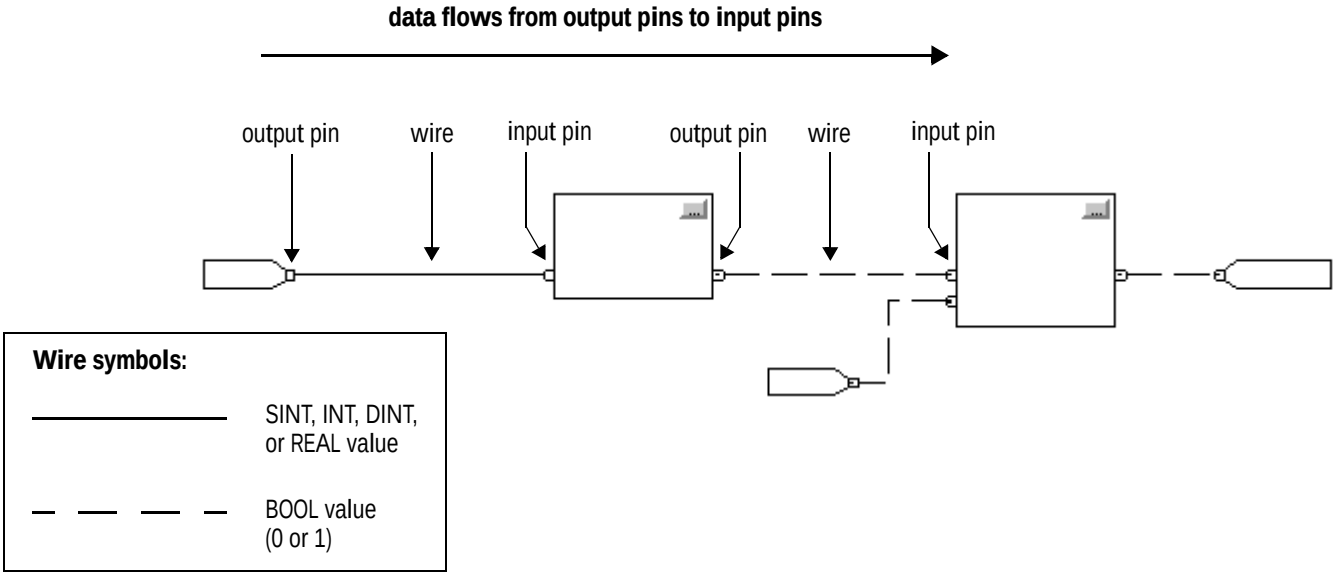
z is the location of the element in the third dimension.

For a structure within a structure, add an additional *member_name*.

TIP I/O module data updates asynchronously to the execution of logic. If you reference an input multiple times in your logic, the input could change state between separate references. If you need the input to have the same state for each reference, buffer the input value and reference that buffer tag. For more information on buffering data, see page28.

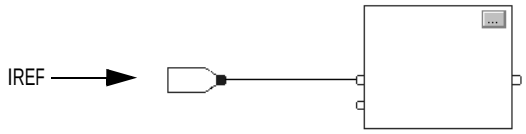
Define the Order of Execution

You define execution order (flow of data) by wiring elements together and indicating any input (feedback) wires, if necessary. The location of a block does not affect the order in which the blocks execute.

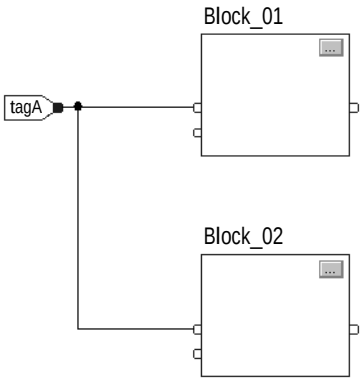


Data Latching

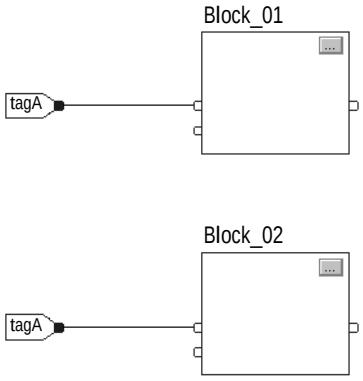
If you use an IREF to specify input data for a Function Block instruction, the data in that IREF is latched for the scan of the Function Block routine. The IREF latches data from program-scoped and controller-scoped tags. The controller updates all IREF data at the beginning of each scan.



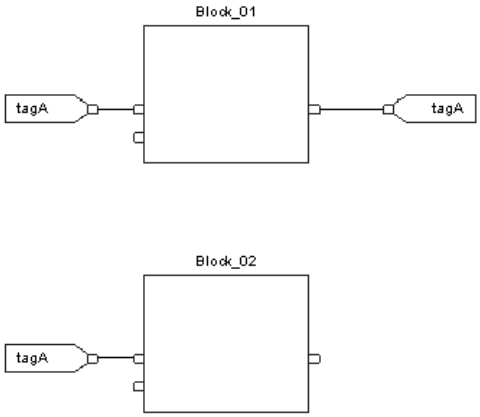
In this example, the value of tagA is stored at the beginning of the routine's execution. The stored value is used when Block_01 executes. The same stored value is also used when Blcock_02 executes. If the value of tagA changes during execution of the routine, the stored value of tagA in the IREF does not change until the next execution of the routine.



This example is the same as the one above. The value of tagA is stored only once at the beginning of the routine's execution. The routine uses this stored value throughout the routine.



Starting with RSLogix 5000 software, version 11, you can use the same tag in multiple IREFs and an OREF in the same routine. Because the values of tags in IREFs are latched every scan through the routine, all IREFs will use the same value, even if an OREF obtains a different tag value during execution of the routine. In this example, if tagA has a value of 25.4 when the routine starts executing this scan, and Block_01 changes the value of tagA to 50.9, the second IREF wired into Block_02 will still use a value of 25.4 when Block_02 executes this scan. The new tagA value of 50.9 will not be used by any IREFs in this routine until the start of the next scan.



Order of Execution

The RSLogix 5000 programming software automatically determines the order of execution for the Function Blocks in a routine when you:

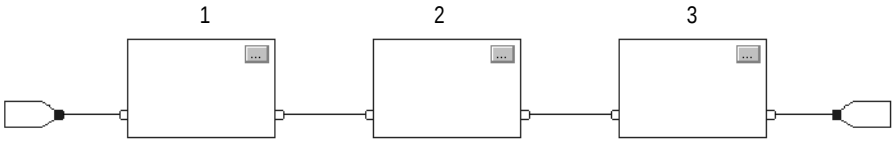
- verify a Function Block routine
- verify a project that contains a Function Block routine
- download a project that contains a Function Block routine

You define execution order by wiring Function Blocks together and indicating the data flow of any feedback wires, if necessary.

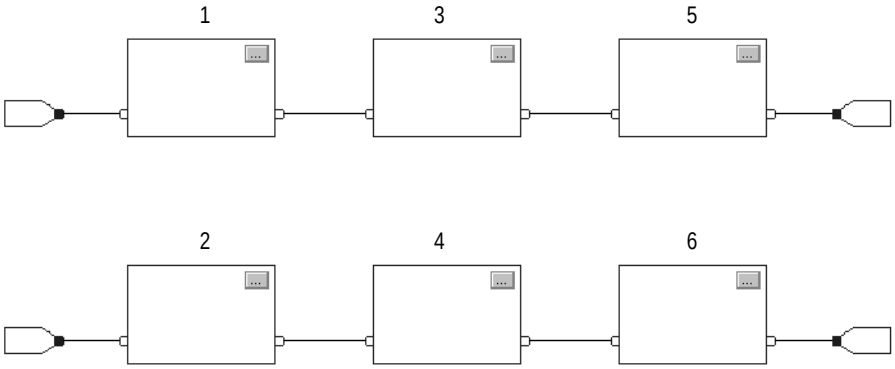
If Function Blocks are not wired together, it does not matter which block executes first. There is no data flow between the blocks.



If you wire the blocks sequentially, the execution order moves from input to output. The inputs of a block require data to be available before the controller can execute that block. For example, block 2 has to execute before block 3 because the outputs of block 2 feed the inputs of block 3.

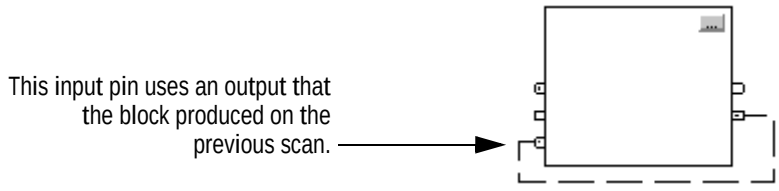


Execution order is only relative to the blocks that are wired together. This example is fine because the two groups of blocks are not wired together. The blocks within a specific group execute in the appropriate order in relation to the blocks in that group.

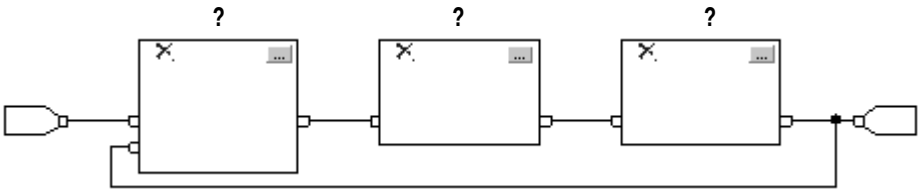


Resolve a Loop

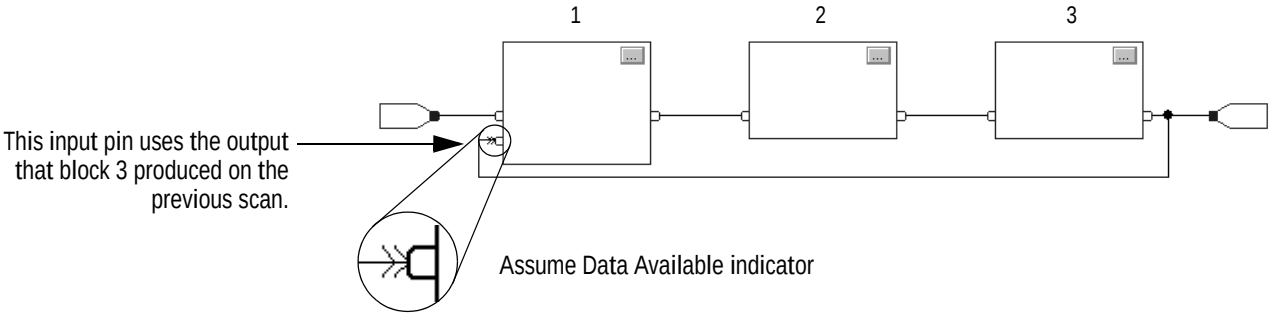
To create a feedback loop around a block, wire an output pin of the block to an input pin of the same block. This example is fine. The loop contains only a single block, so execution order does not matter.



If a group of blocks are in a loop, the controller cannot determine which block to execute first. In other words, it cannot resolve the loop.

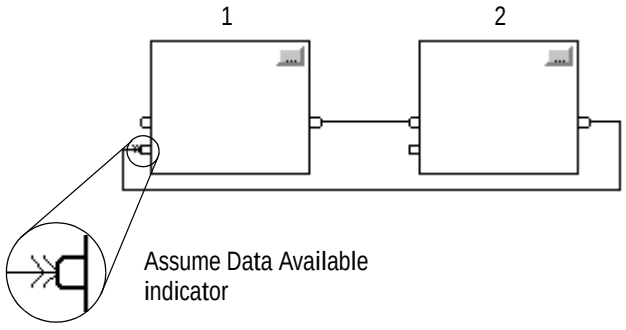
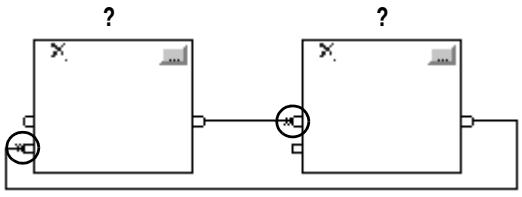


To identify which block to execute first, mark the input wire that creates the loop (the feedback wire) with the Assume Data Available indicator. In this example, block 1 uses the output from block 3 that was produced in the previous execution of the routine.



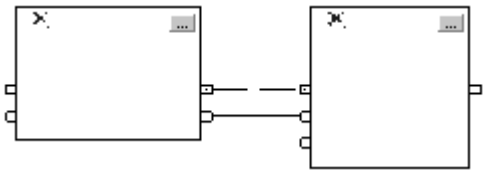
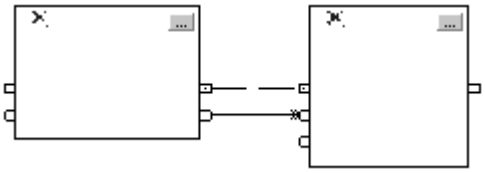
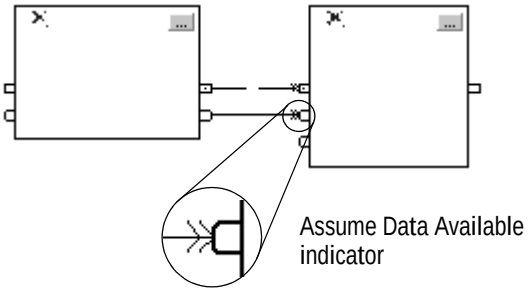
The Assume Data Available indicator defines the data flow within the loop. The arrow indicates that the data serves as input to the first block in the loop.

Do not mark all the wires of a loop with the Assume Data Available indicator.

This is OK	This is NOT OK
 The Assume Data Available indicator defines the data flow within the loop.	 The controller cannot resolve the loop because all the wires use the Assume Data Available indicator.

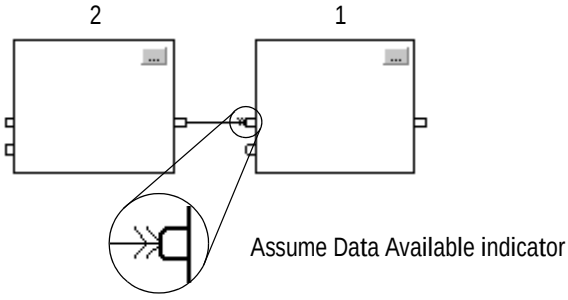
Resolve Data Flow Between Two Blocks

If you use two or more wires to connect two blocks, use the same data flow indicators for all of the wires between the two blocks.

This is OK	This is NOT OK
 Neither wire uses the Assume Data Available indicator.	 One wire uses the Assume Data Available indicator while the other wire does not.
 Both wires use the Assume Data Available indicator.	

Create a One Scan Delay

To produce a one scan delay between blocks, use the Assume Data Available indicator. In this example, block 1 executes first. It uses the output from block 2 that was produced in the previous scan of the routine.



Summary

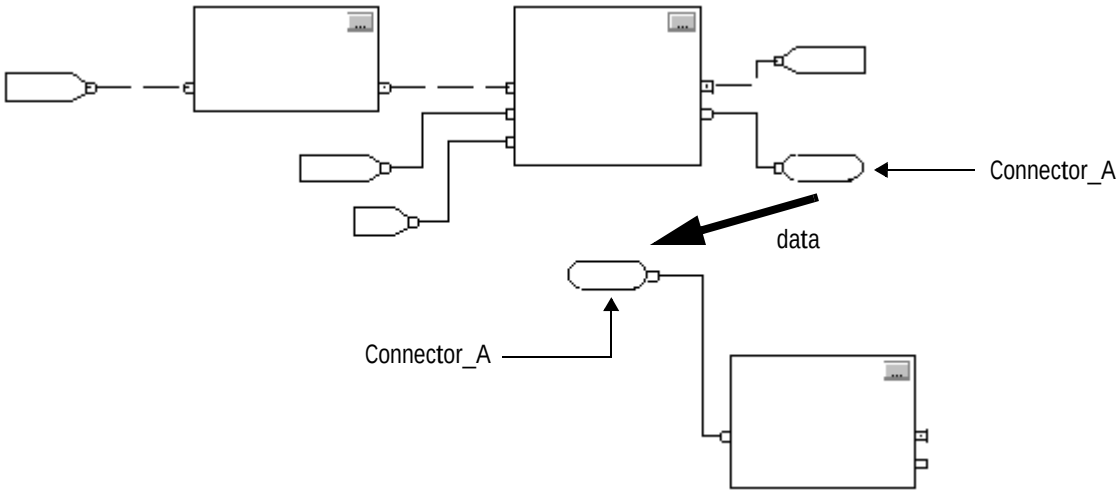
A Function Block routine executes in this order:

1. The controller latches all data values in IREFs.
2. The controller executes the other Function Blocks in the order determined by how they are wired.
3. The controller writes outputs in OREFs.

Identify any Connectors

Like wires, connectors transfer data from output pins to input pins. Use connectors when:

- the elements that you want to connect are on different sheets within the same routine
- a wire is difficult to route around other wires or elements
- you want to disperse data to several points in the routine



To use connectors, follow these rules:

- Each OCON requires a unique name.
- For each OCON, you must have at least one corresponding ICON (i.e., an ICON with the same name as the OCON).
- Multiple ICONs can reference the same OCON. This lets you disperse data to several points in your routine.

Define Program/Operator Control

Several instructions support the concept of Program/Operator control. These instructions include:

- Enhanced Select (ESEL)
- Totalizer (TOT)
- Enhanced PID (PIDE)
- Ramp/Soak (RMPS)
- Discrete 2-State Device (D2SD)
- Discrete 3-State Device (D3SD)

Program/Operator control lets you control these instructions simultaneously from both your user program and from an operator interface device. When in Program control, the instruction is controlled by the Program inputs to the instruction; when in Operator control, the instruction is controlled by the Operator inputs to the instruction.

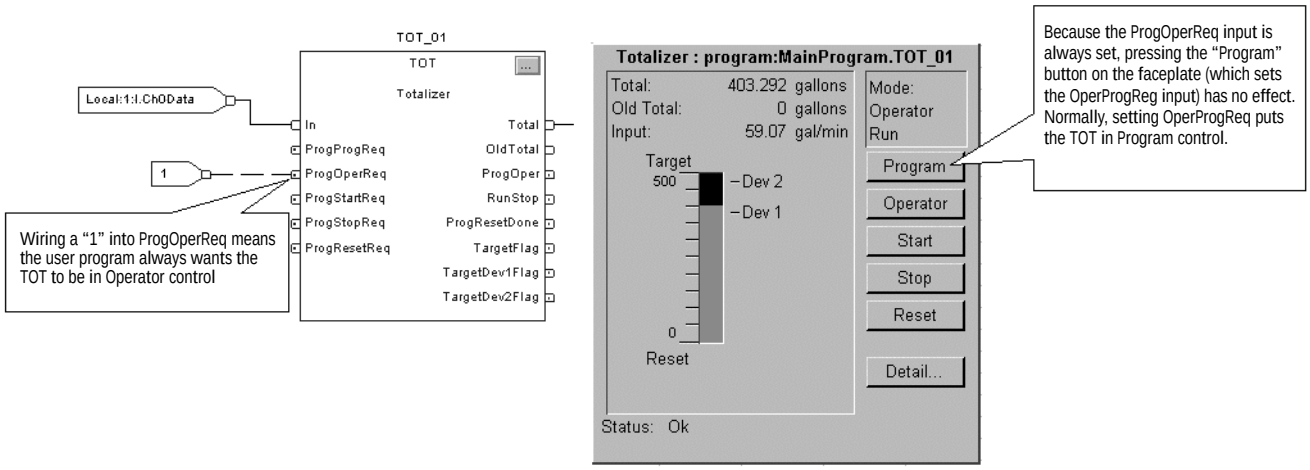
Program or Operator control is determined by using these inputs:

Input:	Description:
.ProgProgReq	A program request to go to Program control.
.ProgOperReq	A program request to go to Operator control.
.OperProgReq	An operator request to go to Program control.
.OperOperReq	An operator request to go to Operator control.

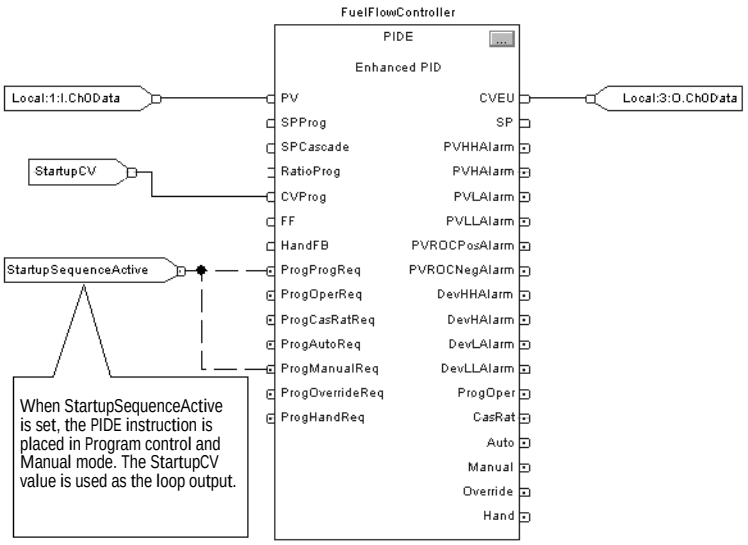
To determine whether an instruction is in Program or Control control, examine the ProgOper output. If ProgOper is set, the instruction is in Program control; if ProgOper is cleared, the instruction is in Operator control.

Operator control takes precedence over Program control if both input request bits are set. For example, if ProgProgReq and ProgOperReq are both set, the instruction goes to Operator control.

The Program request inputs take precedence over the Operator request inputs. This provides the capability to use the ProgProgReq and ProgOperReq inputs to "lock" an instruction in a desired control. For example, let's assume that a Totalizer instruction will always be used in Operator control, and your user program will never control the running or stopping of the Totalizer. In this case, you could wire a literal value of 1 into the ProgOperReq. This would prevent the operator from ever putting the Totalizer into Program control by setting the OperProgReq from an operator interface device.



Likewise, constantly setting the ProgProgReq can "lock" the instruction into Program control. This is useful for automatic startup sequences when you want the program to control the action of the instruction without worrying about an operator inadvertently taking control of the instruction. In this example, you have the program set the ProgProgReq input during the startup, and then clear the ProgProgReq input once the startup was complete. Once the ProgProgReq input is cleared, the instruction remains in Program control until it receives a request to change. For example, the operator could set the OperOperReq input from a faceplate to take over control of that instruction. This example shows how to lock an instruction into Program control.



Operator request inputs to an instruction are always cleared by the instruction when it executes. This lets operator interfaces to work with these instructions by merely setting the desired mode request bit. You don't have to program the operator interface to reset the request bits. For example, if an operator interface sets the OperAutoReq input to a PIDE instruction, when the PIDE instruction executes, it determines what the appropriate response should be and clears the OperAutoReq.

Program request inputs are not normally cleared by the instruction because these are normally wired as inputs into the instruction. If the instruction clears these inputs, the input would just get set again by the wired input. There might be situations where you want to use other logic to set the Program requests in such a manner that you want the Program requests to be cleared by the instruction. In this case, you can set the ProgValueReset input and the instruction will always clear the Program mode request inputs when it executes.

In this example, a rung of ladder logic in another routine is used to one-shot latch a ProgAutoReq to a PIDE instruction when a pushbutton is pushed. Because the PIDE instruction automatically clears the Program mode requests, you don't have to write any ladder logic to clear the ProgAutoReq after the routine executes, and the PIDE instruction will receive only one request to go to Auto every time the pushbutton is pressed.

When the TIC101AutoReq Pushbutton is pressed, one-shot latch ProgAutoReq for the PIDE instruction TIC101. TIC101 has been configured with the ProgValueReset input set, so when the PIDE instruction executes, it automatically clears ProgAutoReq.



Add a Sheet

To add a sheet to a Function Block routine:

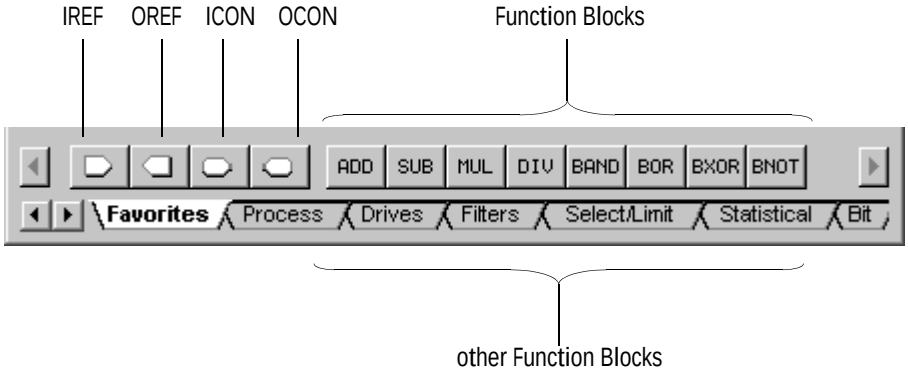
1. Click 



2. Type a description of the sheet (up to 50 characters).

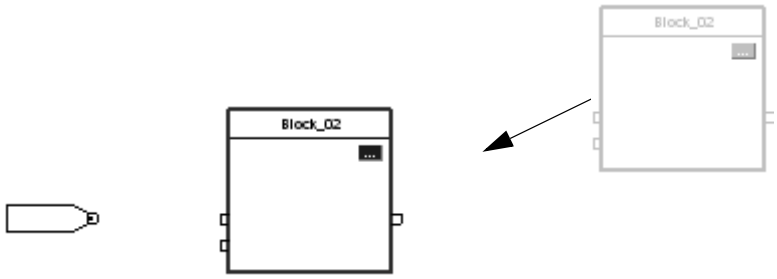
Add a Function Block Element

Use the Language Element toolbar to add a Function Block to a routine.

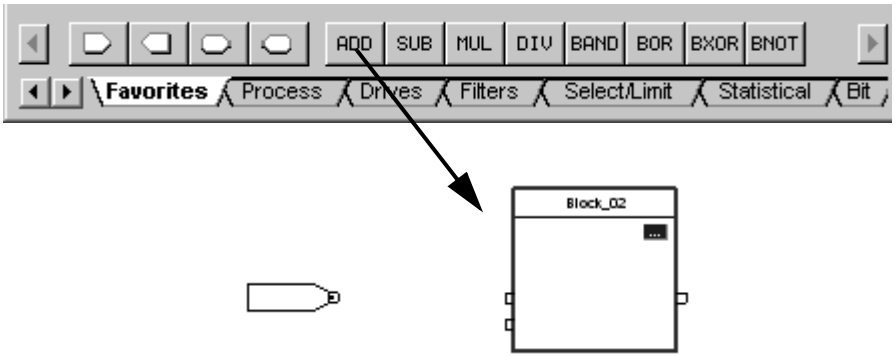


To add an element:

1. On the Language Element toolbar, click the button for the element that you want to add.
2. Drag the element to the desired location.



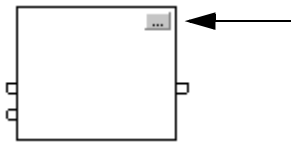
You can also drag the button for the element directly to the desired location.



Connect Elements

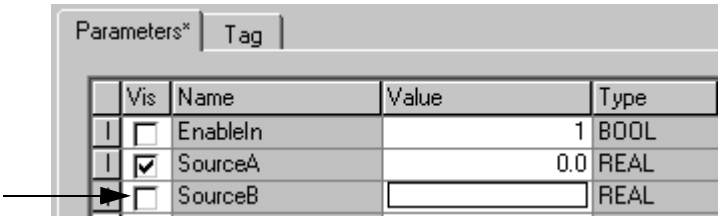
Show or Hide a Pin

When you add a Function Block instruction, the block appears with a set of pins for the default parameters. The rest of the pins are hidden. To show or hide a pin:



1. Click the ... button of the block.
2. Clear or check the Vis check box of the pin:

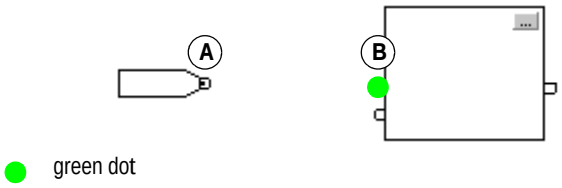
If you want to:	Then;
hide a pin	Clear (uncheck) its Vis check box.
show a pin	Check its Vis check box.



3. Click OK

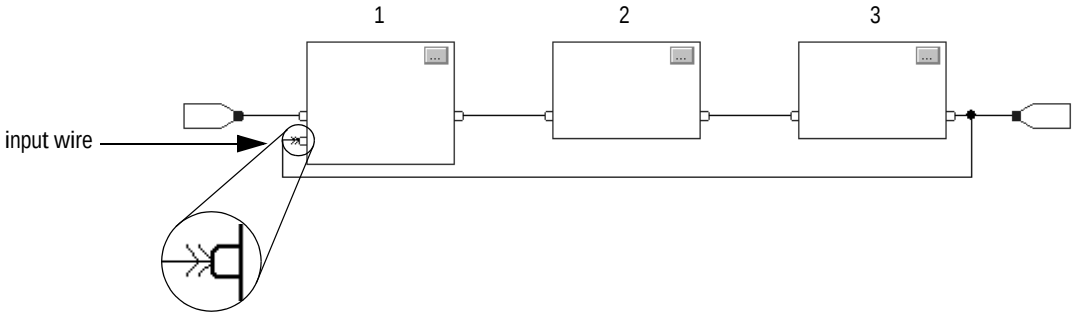
Wire Elements Together

To wire (connect) two elements together, click the output pin of the first element and then click the input pin of the other element. A green dot shows a valid connection point.



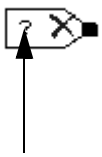
Mark a Wire with the Assume Data Available Indicator

To define a wire as an input, right-click the wire and choose Assume Data Available.



Assign a Tag

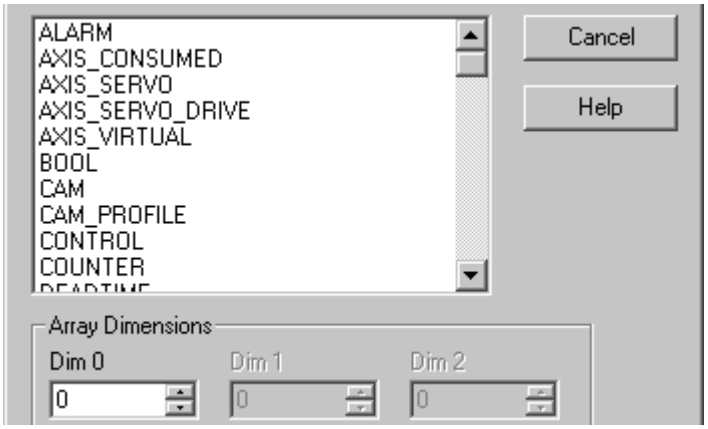
Create and Assign a New Tag



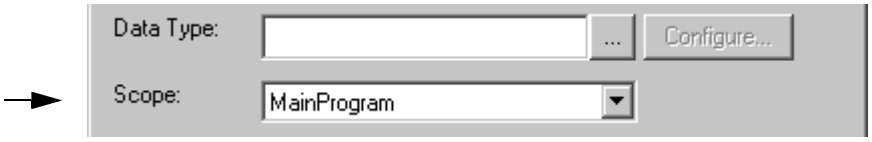
1. Double-click the operand area.
2. Type a name for the tag and press [Enter].
3. Right-click the tag name and choose New “tag_name”.



4. Click the [...] button.

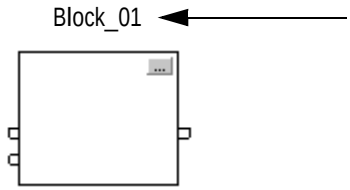


5. Select the data type for the tag.
6. If the tag as an array, type the number of elements in each dimension.
7. Click [OK]
8. Select the scope for the tag.



9. Click [OK]

Assign an Existing Tag



1. Double-click the operand area.
2. Click the t
3. Select the tag:

To select a:	Do this:
tag	Double-click the tag name.
bit number	A. Click the tag name. B. To the right of the tag name, click t C. Click the required bit.

4. Press [Enter] or click a different spot on the diagram.

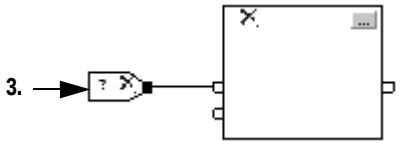
Assign an Immediate Value (Constant)

To assign a constant value instead of a tag value to an input parameter, you have these options:

If you want to:	Then:
make the value visible on the diagram and reports	Use an IREF
be able to change the value online without editing the routine	Enter a Value in the Tag of a Block

Use an IREF

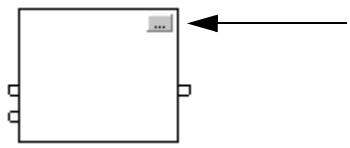
1. Add an IREF.
2. Wire the IREF to the input pin that gets the value.



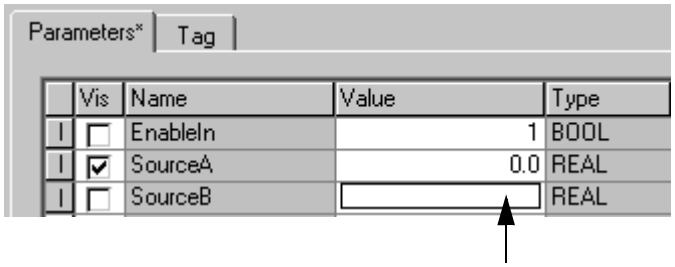
3. Double-click the operand area of the IREF.
4. Type the value and press [Enter].

Enter a Value in the Tag of a Block

To assign a value to a parameter when on wire connects to its pin:



1. Click the [...] button of the block.

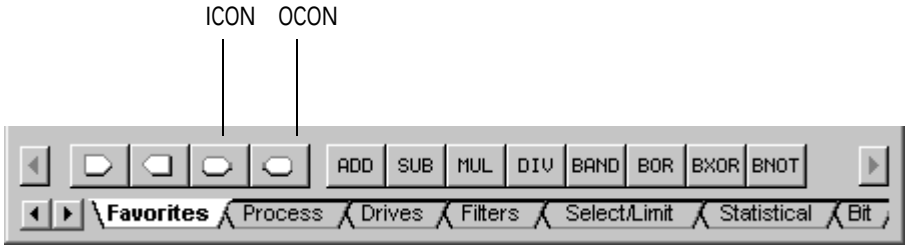


2. Type the value.

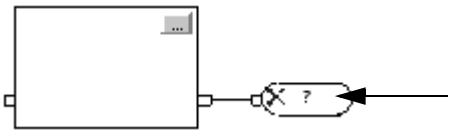
3. Click [OK]

Connect Blocks with an OCON and ICON

Use an OCON or ICON to transfer data between sheets or in complex wiring situations:



Add an OCON



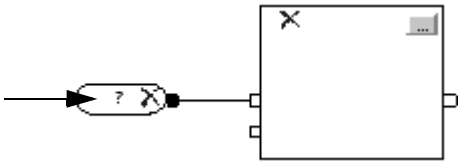
1. Add an output wire connector (OCON) and place it near the output pin that supplies the value.

2. Wire the OCON to the output pin.

3. Double-click the operand area of the OCON.

4. Type a name that identifies the connector and press [Enter].

Add an ICON



1. Add an input wire connector (ICON) and place it near the input pin that gets the value from the corresponding OCON.
2. Wire the ICON to the input pin.
3. Double-click the operand area of the ICON.
4. Select the name of the OCON that supplies the value to this connector and then click a blank spot on the diagram

Verify the Routine

As you program your routine, periodically verify your work:

1. In the top-most toolbar of the RSLogix 5000 window, click 
2. If any errors are listed at the bottom of the window:
   a. To go to the first error or warning, press [F4].
   b. Correct the error according to the description in the Results window.
   c. Go to step 1.
3. To close the Results window, press [Alt] + [1].

Designing Add-On Instructions

Introduction

This chapter discusses the ability for you to maximize program design reuse by using the Add-On Instruction function in RSLogix 5000 software.

Topic	Page
What is an Add-On Instruction?	254
What are the Parts of an Add-On Instruction Definition?	255
Deciding to Develop an Add-On Instruction	259
Available Languages	259
Nesting Add-On Instructions	260
Unavailable Instructions within Add-On Instructions	263
Planning the Design of an Add-On Instruction	265
Developing an Add-On Instruction	265
Creating an Add-On Instruction	267
Considerations when Selecting a Language for an Add-On Instruction	268
Create Parameters	270
Determining Which Parameters to Make Visible or Required	274
Create Local Tags	276
Enter the Logic of the Add-On Instruction	279
Optimizing Performance Considerations	279
Defining the Operation of an Instruction in Different Scan Modes	280
Enabling Scan Modes	281
Prescan Routine	281
Postscan Routine	283
EnableInFalse Routine	285
Using the EnableIn and EnableOut Parameters	287
Testing the Add-On Instruction	289
Managing Add-On Instruction Definitions	292
Updating an Add-On Instruction to a Newer Revision	294
Defining Source Protection for an Add-On Instruction	298
Creating Instruction Help	303

What is an Add-On Instruction?

Add-On Instructions are available beginning with RSLogix 5000 version 16 software. Add-On Instructions are custom instructions that you design and create. With Add-On Instructions, you can create new instructions for sets of commonly-used logic, provide a common interface to this logic, and provide documentation for the instruction.

Add-On Instructions are intended to be used to encapsulate commonly used functions or device control. They are not intended to be a high-level hierarchical design tool. Programs with routines are better suited to contain code for the area or unit levels of your application.

These are some benefits to using an Add-On Instruction.

- Reuse code
  - You can use Add-On Instructions to promote consistency between projects by reusing commonly-used control algorithms.
  - If you have an algorithm that will be used multiple times in the same project or across multiple projects, it may make sense to incorporate that code inside an Add-On Instruction to make it modular and easier to reuse.
- Provide an easier to understand interface
  - You can place complicated algorithms inside of an Add-On Instruction, and then provide an easier to understand interface by making only essential Parameters visible or required.
  - You can reduce documentation development time through automatically generating instruction help.
- Protection of intellectual property
  - You can place your proprietary code inside of an Add-On Instruction, then use Source Protection to prevent others from viewing or changing your code.
- Simplify maintenance
  - You can simplify code maintenance since Add-On Instruction logic animates for a single instance.

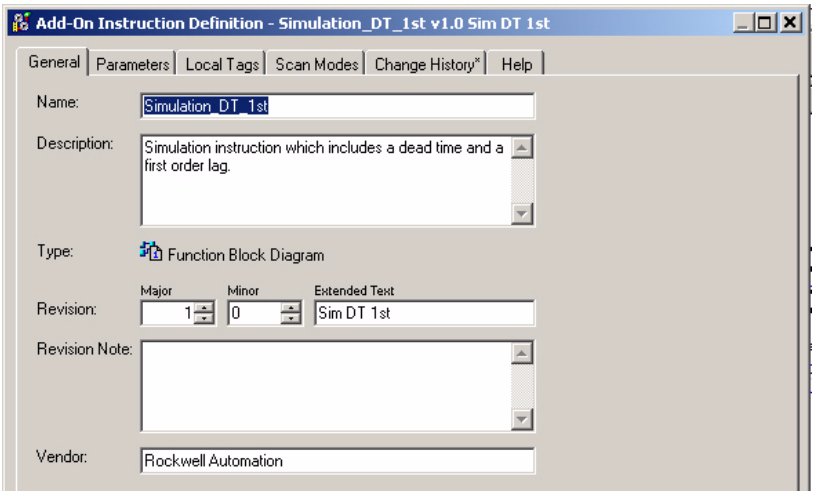
An Add-On-Instruction can be used across multiple projects. You can define the instructions, or the instructions can be provided to you by someone else, or they can be copied from another project.

Once defined in a project, they behave similarly to the built-in instructions already available in the RSLogix 5000 software. They appear on the instruction toolbar and in the instruction browser for easy access, just like built-in RSLogix 5000 software instructions.

What are the Parts of an Add-On Instruction Definition?

Add-On Instructions are made up of the following parts.

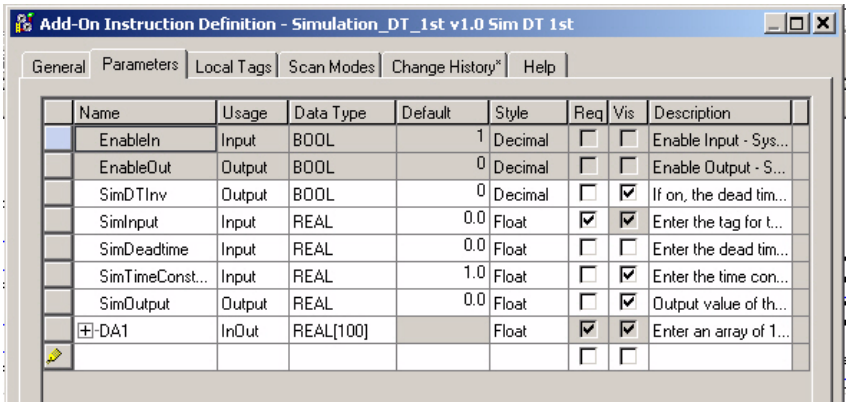
General Information



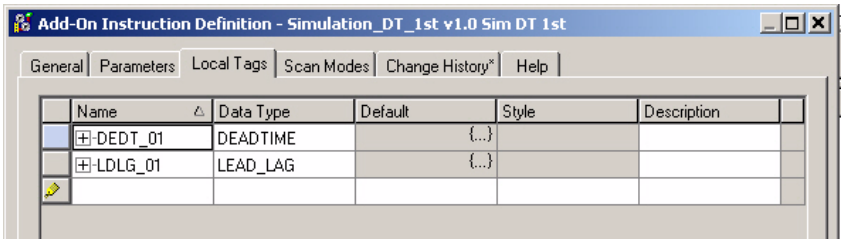
The General tab contains the information from when you first created the instruction. You can use this tab to update the information. The description, revision, revision note, and vendor information is copied into the custom help for the instruction. The revision is not automatically managed by the software. You are responsible in defining how it is used and when it is updated.

Parameters

The Parameters define the instruction interface; how the instruction appears when used. The Parameter order defines the order that the Parameters appear on the instruction call.



Local Tags



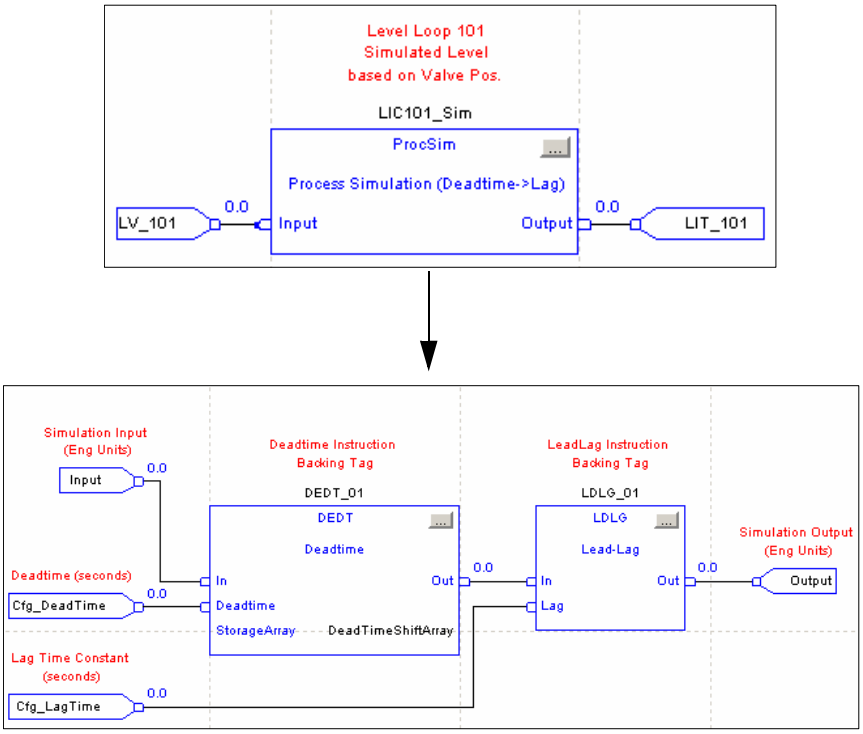
Local Tags are hidden members and are not visible outside the instruction and cannot be referenced by other programs or routines. They are private to the instruction.

Data Type

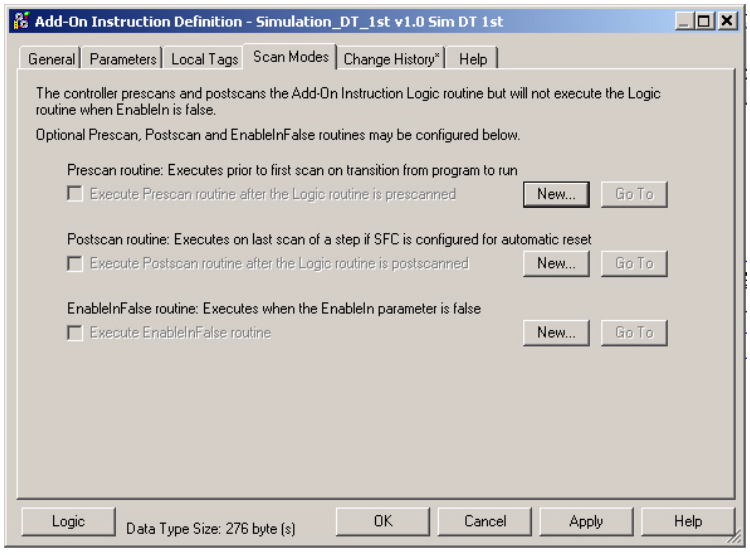
Parameters and Local Tags are used to define the data type that is used when executing the instruction. The software builds the associated data type. The software orders the members of the data type that correspond to the Parameters in the order that the Parameters are defined. Local Tags are added as hidden members.

Logic Routine

The Logic routine of the Add-On Instruction defines the primary functionality of the instruction. It is the code that executes whenever the instruction is called. Shown below is the interface of an Add-On Instruction and its primary Logic routine that defines what the instruction does.



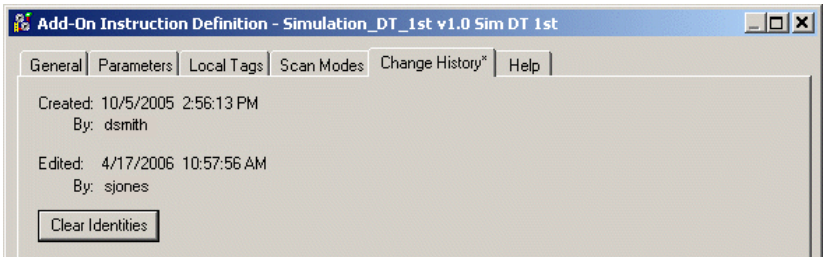
Optional Scan Mode Routines



You can define additional routines for Scan mode behavior.

Refer to Defining the Operation of an Instruction in Different Scan Modes on page 280 for more information.

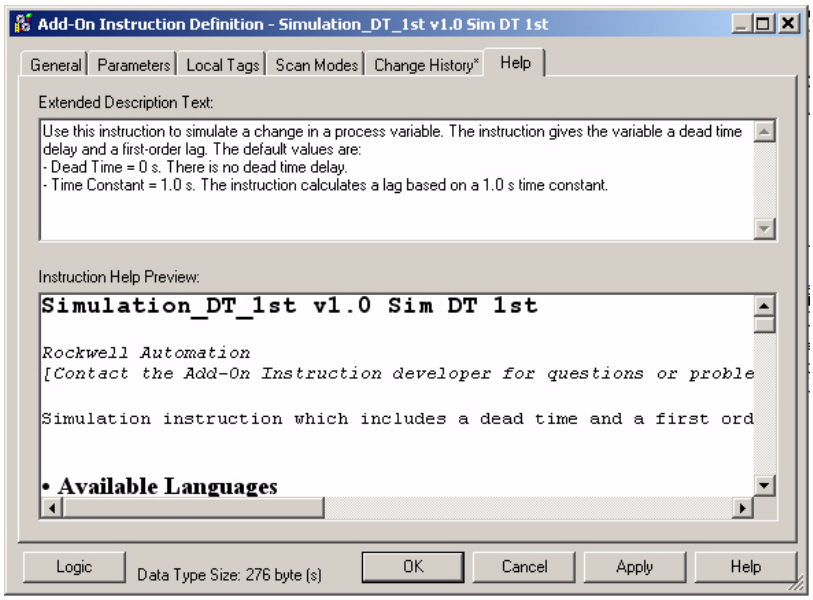
Change History



The Change History tab displays the creation and latest edit information that is tracked by the software. The By fields show who made the change based on the Windows user name at the time of the change.

Help

The Name, Revision, Description, and Parameter definitions are used to automatically build the Instruction help. Use the Extended Description Text to provide additional Help documentation for the Add-On Instruction. The Instruction Help Preview shows how your instruction will appear in the various languages, based on Parameters defined as Required or Visible.



Deciding to Develop an Add-On Instruction

When deciding whether to develop an Add-On Instruction, consider the following aspects.

Instruction Functionality

Complex instructions tend to be highly application specific and not reusable, or require extensive configuration support code. As with the built-in instructions, Add-On Instructions need to do one thing well and support modular coding. Consider how the instruction will be used and manage interface complexity for the end user or application.

Add-On Instructions are best at providing a specific type of functionality or device control.

Encapsulation

Add-On Instructions are designed to fully encapsulate the code and data associated with the instruction. The logic inside an Add-On Instruction only uses the Parameters and Local Tags defined by the instruction definition. There is no direct programmatic access to controller or program scope tags. This lets the Add-On Instruction be a standalone component that can execute in any application that calls it by using the Parameters interface. It can be validated once and then locked to prevent edits.

Available Languages

Add-On Instruction routines have a choice of three RSLogix 5000 software programming languages:

- Ladder Diagram
- Function Block Diagram
- Structured Text

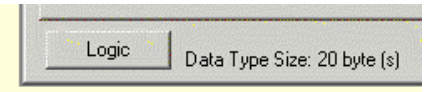
Once created, the instruction can be called from any of the RSLogix 5000 languages. These languages are available in RSLogix 5000 software, depending on what package edition you purchase.

Refer to the RSLogix 5000 Software Product Profile, publication 9324-PP001G for more information.

An Add-On Instruction written in one language can be used as an instruction through a call in another language.

Instruction Size

Add-On Instructions have one primary Logic routine that defines the behavior of the instruction when executed. This Logic routine is like any other routine in the project and has no additional restrictions in length. The total number of Input Parameters plus Output Parameters plus Local Tags can be up to 512. There are no limits on the number of InOut Parameters. The maximum data instance supported (which includes Inputs, Outputs, and Local Tags) is two megabytes. The data type size displays on the bottom of the Parameter and Local Tabs in the Add-On Instruction Definition.



Runtime Editing of Functionality

Add-On Instructions definitions can only be edited offline. If the intended functionality may need to be changed in a running controller, consider carefully if an Add-On Instruction is suitable.

Nesting Add-On Instructions

Add-On Instructions can call other Add-On Instructions in their routines. This provides the ability to design more modular code by creating simpler instructions that can be used to build more complex functionality by nesting instructions. The instructions can be nested to seven levels deep.

Add-On Instructions cannot call other routines via a JSR. You must use a nested instruction if you need complex functionality consisting of multiple routines.

Using a Routine Versus an Add-On Instruction

You can write your code in three basic ways: to run in-line as a main routine, using subroutine calls, or as Add-On Instructions. The following table summarizes how each of these ways has it's advantages and disadvantages.

Aspect	Main Routine	Subroutine	Add-On Instruction
Accessibility	N/A	Within program (multiple copies, one for each program)	Anywhere in controller (single copy for the entire project)
Parameters	N/A	Pass by value	Pass by value via Input and Output Parameter or by reference via InOut Parameter
Numeric Parameters	N/A	No conversion, user must manage	Automatic data type conversion for Input and Output Parameters
Parameters data types	N/A	Atomic, arrays, structures	Atomic for any Parameter Arrays and structures must be an InOut Parameter
Parameter checking	N/A	None, user must manage	Verification checks that correct type of argument has been provided for a Parameter
Data encapsulation	N/A	All data at program or controller scope (accessible to anything)	Local data is isolated (only accessible within instruction)
Monitor/debug	In-line code with its data	Mixed data from multiple calls, which complicates debugging	Single calling instance data, which simplifies debugging
Supported programming languages	FBD, LD, SFC, ST	FBD, LD, SFC, ST	FBD, LD, ST
Callable from	N/A	FBD, LD, SFC, ST	FBD, LD, SFC via ST, ST
Protection	Locked and View Only	Locked and View Only	Locked and View Only
Documentation	Routine, rung, textbox, line	Routine, rung, textbox, line	Instruction, revision info, vendor, rung, textbox, line, extended help
Execution performance	Fastest	JSR/SBR/RTN instructions add overhead All data is copied Indexed reference impact	Call is more efficient InOut Parameters are passed by reference, which is faster than copying data for many types Parameter references are automatically offset from passed in instruction tag location
Memory use	Most used	Very compact	Compact call requires more memory than a subroutine call All references need an additional word
Edit	Online/offline	Online/offline	Offline only
Import/export	Only LD rungs with referenced tags / UDTs	Only LD rungs with referenced tags / UDTs	Full instruction definition including routines and tags to L5X

Programmatic Access to Data

Input and Output Parameters and Local Tags are used to define an instruction-defined data type. Each Parameter or local tag has a member in the data type, although local tag members are hidden from external use. Each call to an Add-On Instruction utilizes a tag of this data type to provide the data instance for the instruction's execution.

The Parameters of an Add-On Instruction are directly accessible in the controller's programming via this instruction-defined tag within the normal tag scoping rules.

The Local Tags are not accessible programmatically through this tag. This has impact on the usage of the Add-On Instruction. If a structured (including UDTs), array, or nested Add-On Instruction type is used as a Local Tag (not InOut Parameters), then they are not programmatically available outside the Add-On Instruction definition.

Unavailable Instructions within Add-On Instructions

Most built-in instructions can be used within Add-On Instructions. The following instructions cannot be used.

Unavailable Instruction	Description
BRK	Break
EOT	End of Transition
EVENT	Event Task Trigger
FOR	For (For/Next Loop)
IOT	Immediate Output
JSR	Jump to Subroutine
JXR	Jump to External Routine
MAOC	Motion Arm Output Cam
PATT	Attach to Equipment Phase
PCLF	Equipment Phase Clear Failure
PCMD	Equipment Phase Command
PDET	Detach from Equipment Phase
POVR	Equipment Phase Override Command
RET	Return
Safety Instructions	Any of the Safety Instructions
SBR	Subroutine
SFP	SFC Pause
SFR	SFC Reset

In addition, the following instructions may be used in an Add-On Instruction, but the data instances must be passed as an InOut Parameter.

Instruction	Description
ALMA	Analog alarm
ALMD	Digital alarm
Motion Instructions	All Motion Instructions
MSG	Message

GSV/SSV Instruction Class Limitations

When using GSV and SSV instructions inside an Add-On Instruction, the supported classes are limited to the following.

- Controller
- Controller Device
- CST
- DF1
- Fault Log
- Program[1]
- Routine[1]
- Redundancy
- Safety
- Serial Port
- Task[1]
- Wall Clock Time

[1] The classes that represent programming components - Task, Program, Routine - only support THIS for the instance.

Planning the Design of an Add-On Instruction

Take time to plan your instruction design. Advance planning can identify problems that you can avoid later. When defining the requirements of an instruction, you are also determining the interface. Keep the following aspects in mind when defining your instruction requirements and design.

Developing an Add-On Instruction

Now that you have decided to develop an Add-On Instruction, next you need to plan the design and then create your new instruction.

Intended Behavior

- What is the purpose for creating the Add-On Instruction?
- What problem is it expected to solve?
- How is it intended to function?

Parameters

- What data needs to be passed to the instruction? What needs to be accessible outside of the instruction? How will the Parameters display?
- Determine which Parameters to make required or just visible. The order of the Parameters defines the appearance of instruction.

Naming Conventions

- What naming conventions are you going to use?
- The instruction name will be used as the mnemonic for your instruction. Although the name can be up to 40 characters long, you will typically want to use shorter, more manageable names.

Source Protection

- What type of source protection needs to be defined, if any?
- Source protection can be used to provide read only access of the Add-On Instruction definition or to completely lock the definition's routines and Local Tags to hide the implementation.

Nesting - Reuse Instructions

- Are there other Add-On Instructions that you can reuse?
- Do you need to design your instructions to share common code?

Local Tags

- What data is needed for your logic to execute but is not public?

- Identify Local Tags you might use in your instruction. Local Tags are useful for items such as intermediate calculation values that you do not want to expose to users of your instruction.

Languages

- What language do you want to use to program your instruction?
- Determine which RSLogix 5000 software programming language to use based on the use and type of application. The primary logic of your instruction will consist of a single routine of code.
- If execution time and memory usage are critical factors, refer to the Logix5000 Controllers Execution Time and Memory Use Reference Manual, publication 1756-RM087.

Scan Mode Routines

- Do you need to provide Scan mode routines?
- You can optionally define the scan behavior of the instruction in different Scan modes. This lets you define unique initialization behaviors on controller startup (Program -> Run), SFC step postscan, or EnableIn False condition.
- Decide on what language any Scan mode routines will be written.

Test

- How will you test the operation of your Add-On Instruction before commissioning it?
- What possible unexpected inputs could the instruction receive, and how will the instruction handle these cases?

Help Documentation

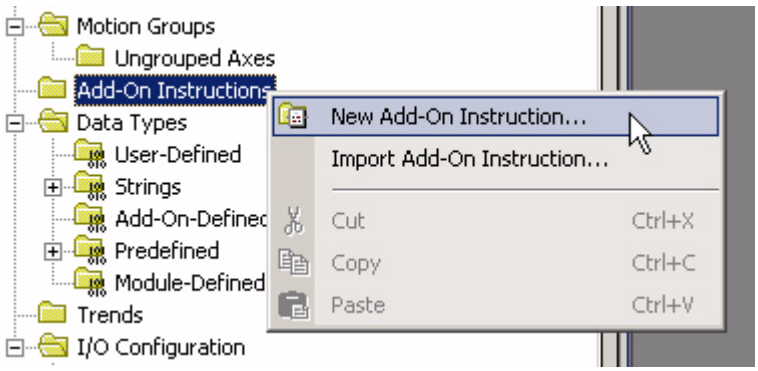
- What information needs to be in the instruction help?
- When you are creating an instruction you have the opportunity to enter information into various description fields. You will also need to develop information on how to use the instruction and how it operates.

Creating an Add-On Instruction

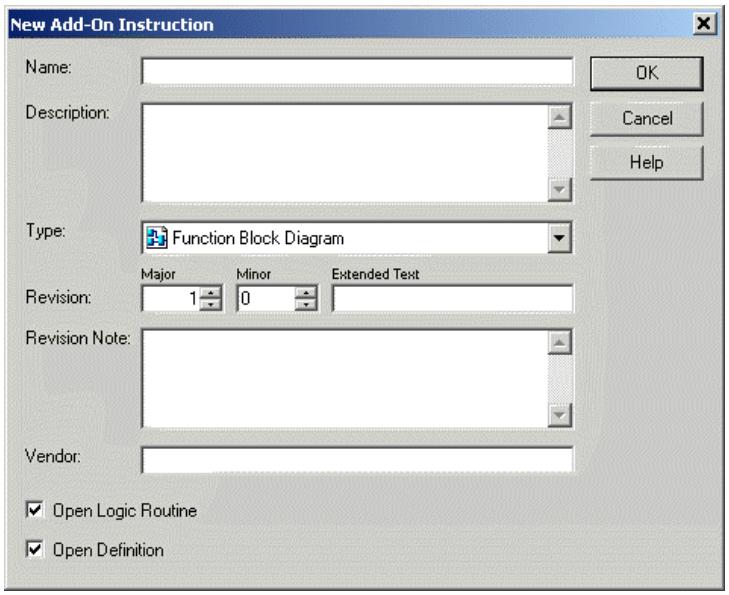
Follow these steps to create a new Add-On Instruction.

1. Open an existing or new project.

2. Right-click the Add-On Instructions folder in the Controller Organizer and select New Add-On Instruction.



The New Add-On Instruction dialog displays.



3. Enter a name for the new instruction.

TIP The name can be up to 40 characters long. It must start with an alpha or underscore, and must contain only alphas, numerics, or underscores. The name must also not match the name of a built-in instruction or an existing Add-On Instruction in your project.

4. Enter a description for the new instruction, maximum 120 characters.

5. Choose a programming language for the Add-On Instruction Logic.

TIP Once you pick a language for an Add-On Instruction, it can't be changed. You must delete an instruction to change the language of the Add-On Instruction Logic routine.

6. Assign a Revision level for the instruction.

7. Write a Revision Note, if necessary.

8. Write a note about the Vendor, if necessary.

9. Click OK to create the instruction.

Considerations when Selecting a Language for an Add-On Instruction

Before you create an instruction you need to decide on what programming language to use. You need to select the language type based on the type of application you are developing. Ladder Diagram, Function Block Diagram, and Structured Text can be used for Add-On Instruction logic.

Each of the programming languages supported in RSLogix 5000 software is targeted for different types of applications and programming styles. In general, Ladder Diagram will execute simple boolean logic, timers, and counters the fastest. Function Block Diagrams and Structured Text may be more efficient if you take advantage of the more advanced process and drives instructions available in those languages.

You cannot compare execution times for the same Add-On Instruction written in different programming languages. There are fundamental differences on how the different languages execute and are compiled.

Transitional Instructions

Some instructions only execute (or retrigger) when rung-condition-in toggles from false to true. These are transitional-relay Ladder instructions. When used in an Add-On Instruction, these instructions will not detect the rung-in transition to the false state. When the EnableIn bit is false, the Add-On Instruction Logic routine no longer executes and thus the transitional instruction does not detect the transition to the false state. Extra conditional logic will be required to handle triggering of transitional instructions contained in an Add-On Instruction.

Some examples of transitional instructions include: ONS, MSG, PXRQ, SRT, some of the ASCII instructions, and some of the Motion instructions.

TIP

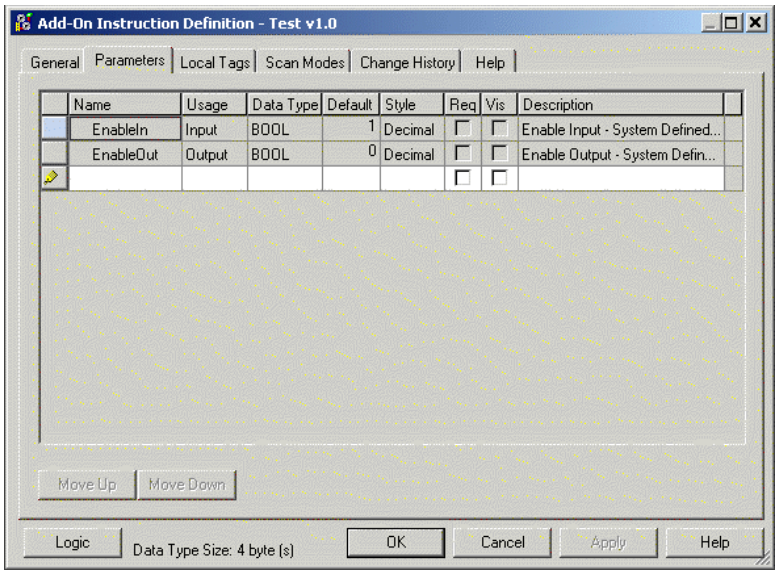
The EnableInFalse routine can be used to provide the conditioning required to retrigger transitional instructions contained in an Add-On Instruction. However, this method will not work for calls to this Add-On Instruction contained in a Structured Text routine, since EnableIn is always true for calls in Structured Text.

Create Parameters

Use the Add-On Instruction Definition Editor to create the Parameters for your instructions. Follow these steps to define the Parameters for your instruction.

1. In the Controller Organizer, right-click an Add-On Instruction and select Open Definition.

2. Click the Parameter tab and enter a name for a Parameter.



3. Define the Usage, based on the type of Parameters: Input, Output, InOut.

4. Select a data type.

   Add-On Instruction Parameter usage and data type options are as follows:

   - Input Parameter - passed by value into the Add-On Instruction. Must be a SINT, INT, DINT, REAL, or BOOL data type.
   - Output Parameter - passed by value out of the Add-On Instruction. Must be a SINT, INT, DINT, REAL, or BOOL data type.
   - InOut Parameter - passed by reference into the Add-On Instruction. It can be any data type including structures and arrays.

TIP An instruction with only input Parameters, except EnableOut, is treated as an input instruction in a Ladder diagram. The EnableOut parameter is used for the rung condition.

5. Set the default values.

   Default values are loaded from the Add-On Instruction definition into the tag of the Add-On Instruction data type when it is created, and anytime a new input or output parameter is added from the Instruction Definition editor to the Add-On Instruction definition.

   TIP One way to be sure that a value is applied to all instances of your instruction is to assign a literal value to the parameter in your code for your Add-On Instruction.

6. Set the display style.

7. Write a description, maximum 120 characters.

   This description appears in the instruction's help.

8. Determine whether the Parameter should be set to Required or Visible.

   Refer to Determining Which Parameters to Make Visible or Required on page 274.

   If you decide to make the Parameter required, it will also be visible.

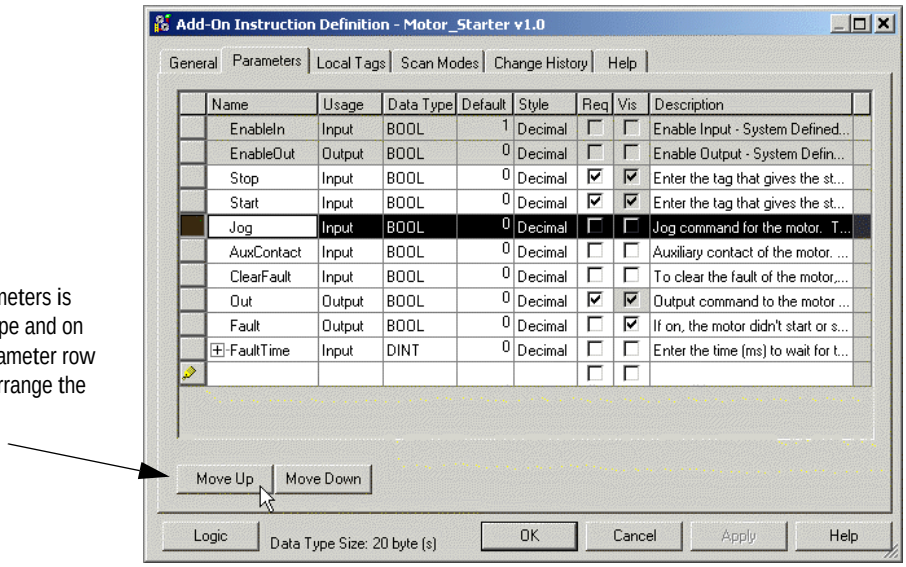
9. Repeat for additional Parameters.

TIP You can also create Parameters using the Tag Editor, New Parameter or Local Tag dialog, or by right-clicking on tag name in the logic of your routine.

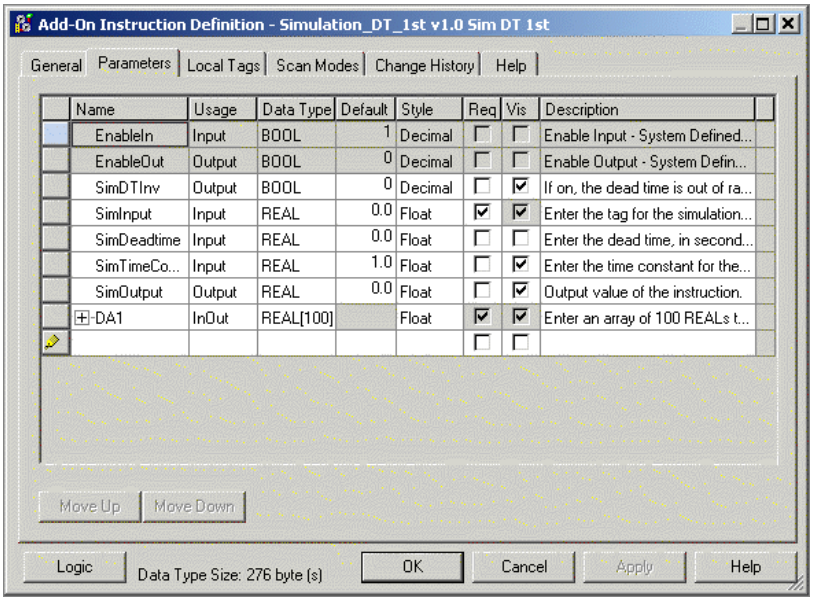
The following are examples of the Parameters tab of the Add-On Instruction Definition Editor for two different instructions.

Motor Starter Parameters Tab

The order that you create the Parameters is how they will appear in the data type and on the instruction face. Select the Parameter row and use Move Up and Down to rearrange the order.



Deadtime and Lag Simulation Parameters Tab



Considerations When Passing Arguments to Parameters by Reference or by Value

The following information will help you understand the differences between passing argument tags to Parameters by reference or by value.

Aspect	By Value (Input or Output)	By Reference (InOut)
Value	Synchronous - the Parameter's value does not change during Add-On Instruction execution.	Asynchronous- the Parameter's value may change during Add-On Instruction execution. Any access by the instruction's logic directly reads or writes the passed tag's value.
Performance	Argument values are copied in and out of the Parameters of the Add-On Instruction. This takes more time to execute a call to the instruction.	Parameters access argument tags directly by reference, which leads to faster execution of instruction calls.
Memory usage	Most amount.	Least amount.
Parameter data types supported	Atomic.	Atomic, Arrays, and structures.

Considerations for Selecting a Data Type for a Parameter

The Logix5000 controllers perform DINT (32 bit) and REAL (32 bit) math operations, which causes DINT data types to execute faster than other integer data types. Data conversion rules of SINT to INT to DINT are applied automatically, and can add overhead. Whenever possible, use DINT data types for the Add-On Instruction input and output Parameters.

Consideration when Using Single Dimension Array as InOut Parameter

The InOut Parameter can be defined to be a single dimension array. When specifying the size of this array consider the following:

- The user of your instruction can pass an array tag that is the same size as your definition or,
- They can pass an array tag that is larger than your definition.

To accommodate this flexibility in your logic, you can determine the actual size of the referenced array by using the SIZE instruction.

TIP

When you monitor an array InOut Parameter inside of the Logic routine, the parameter definition is used to determine the size of the array. For example, if you define an InOut parameter to be a 10-element array of DINTs and the end user passes in an array of 100 DINTs. Then if you open the Add-On Instruction logic, select the appropriate context for that call, and monitor the array parameter, only 10 elements will be displayed.

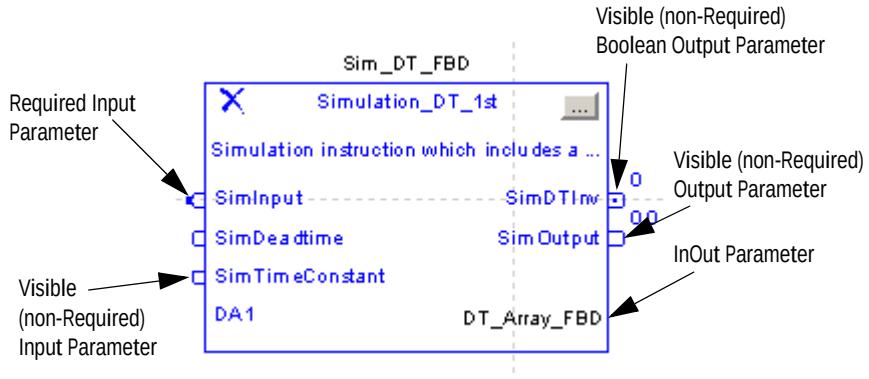
Determining Which Parameters to Make Visible or Required

To help be sure that certain data is passed into the Add-On Instruction, you can use required Parameters. A required Parameter must be passed as an argument in order for a call to the instruction to verify. In Ladder Diagram and Structured Text, this is done by specifying an argument tag for these Parameters. In a Function Block Diagram, required input and output Parameters must be wired, and InOut Parameters must have an argument tag. If a required Parameter does not have an argument associated, as described above, then the routine containing the call to the Add-On Instruction will not verify.

For output Parameters, making a Parameter visible is useful if you don't usually need to pass the Parameter value out to an argument, but you do want to display its value prominently for troubleshooting.

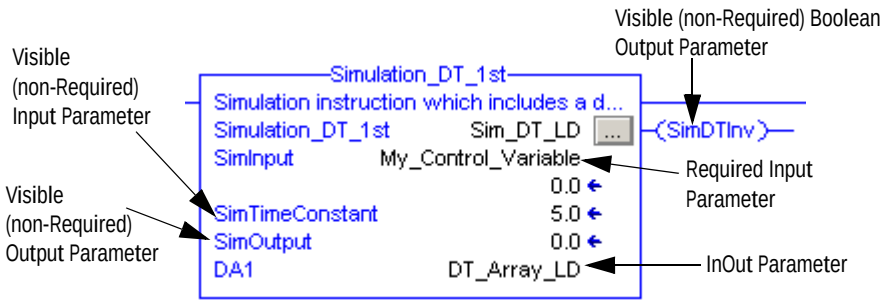
Required Parameters are always visible, and InOut Parameters are always required and visible. All Input and Output Parameters, regardless of being marked as Required or Visible, can be programmatically accessed as a member of the instruction's tag.

Simulation Instruction in Function Block



If you want a pin that is displayed in Function Block, but wiring to it is optional, set it as Visible

Simulation Instruction in Ladder



- If you want the Parameter's value displayed on the instruction face in Ladder, set the Parameter as Visible.
- An Output Parameter of the BOOL tag type that is not required, but visible, will show as a status flag on the right side of the block in Ladder. This can be used for status flags like DN or ER.

Input, Output, and InOut Required and Visible Settings Effects

Type	Required	Visible	Ladder Value	Ladder Argument	FBD Must Connect	FBD Argument	FBD Change Visibility	ST Argument
Input	Y	Y	Y	Y	Y	N/A	N	Y
Input	N	Y	Y	N	N	N/A	Y	N
Input	N	N	N	N	N	N/A	Y	N
Output	Y	Y	Y	Y	Y	N/A	N	Y
Output	N	Y	Y	N	N	N/A	Y	N
Output	N	N	N	N	N	N/A	Y	N
InOut	Y	Y	N	Y	N/A	Y	N	Y

- If you have a Parameter for which the user must specify a tag as its source for input or its destination as output, and you don't want this to be optional, set the Parameter as Required. Any required Parameters are automatically set to Visible.

- The visible setting is always set to visible for InOut Parameters. All InOut Parameters are required.

TIP

When you are using your Add-On Instructions the visible setting may be overridden in Function Block Diagram routines if the Parameter is not required or already wired. Overriding the visibility at the instruction call does not affect this definition configuration.

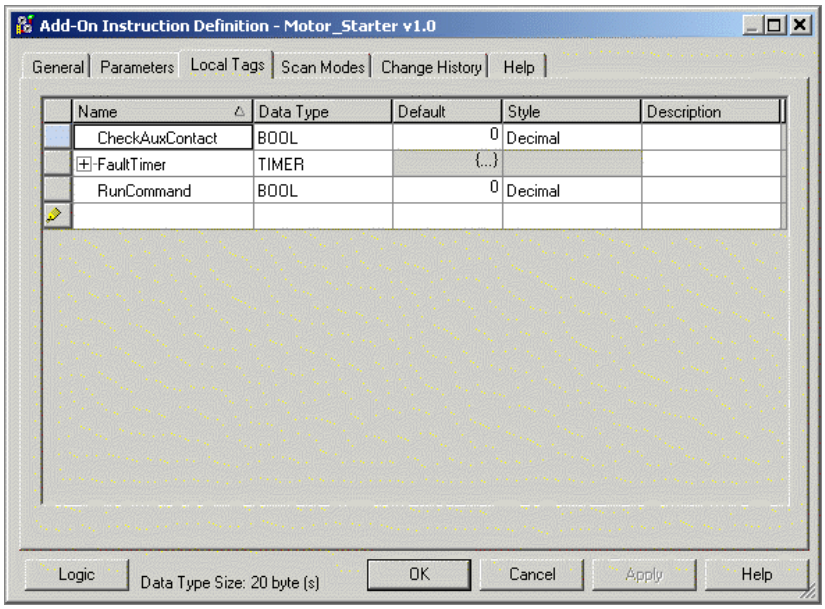
Create Local Tags

Use the Definition Editor to create the Local Tags for your instructions. Local Tags contain data that will be used by your Add-On Instruction but which you do not want exposed to the user of your instruction. Local Tags will not appear in the data structure for an Add-On Instruction because they are hidden members. You can access local tag values from an HMI by specifying the name of the local tag as a member in an Add-On Instruction type tag.
For example, aMotorStarter_Tag.CheckAuxContact.

Follow these steps to define the Local Tags for your instruction.

1. In the Controller Organizer, right-click an instruction and select Open Definition.

2. Click the Local Tags tab and enter a name for a new tag and select a data type.



You cannot use these data types for Local Tags - ALARM_ANALOG, ALARM_DIGITAL, MESSAGE, or any motion data types, for example Axis or MOTION_GROUP. To use these type of tags in your Add-On Instruction, define an InOut Parameter. Local Tags also are limited to single dimension arrays, the same as user data types.

3. Set the default values.

   Default values are loaded from the Add-On Instruction definition into the tag of the Add-On Instruction data type when it is created or any time a new tag is added in the Instruction Definition Editor to the Add-On Instruction definition.

TIP

Local tag values in your Add-On Instruction definition are the default values. Changing an existing local tag value will not affect any instances of your Add-On Instruction that already exist.

One way to be sure a value is applied to all instances of your instruction is to assign a literal value to the tag in your code for your Add-On Instruction.

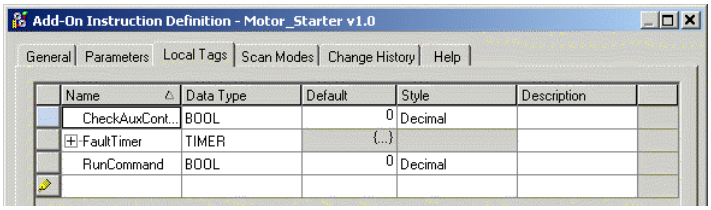
4. Set the display style.

5. Write a description, a maximum of 120 characters.

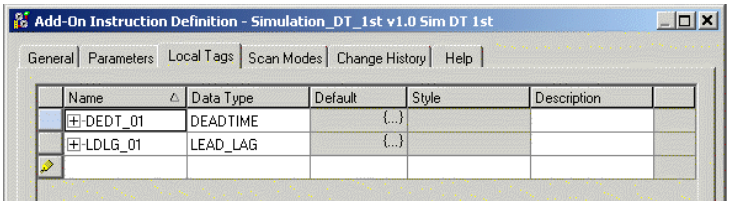
6. Repeat for additional Local Tags.

The following are examples of the Local Tags tab of the Add-On Instruction definition.

Motor Starter Local Tags

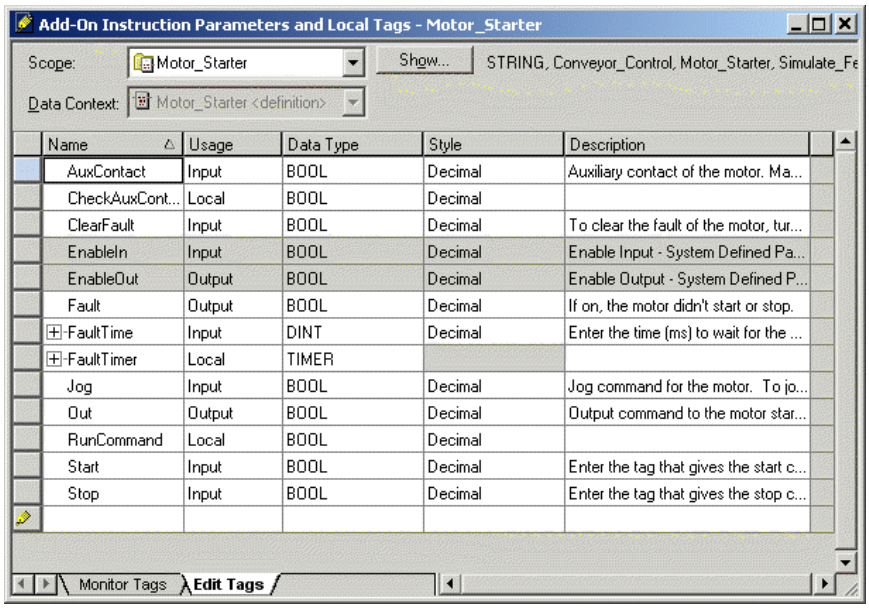


Deadtime and Lag Simulation Local Tags Tab



Motor Starter Tag Editor

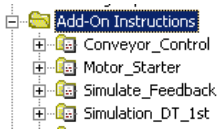
You can also add and edit Parameters and Local Tags using the tag editor as shown below.



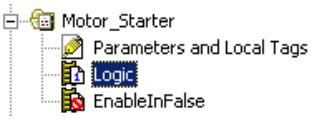
Enter the Logic of the Add-On Instruction

Follow these steps to enter logic into your project.

1. Expand the Add-On Instruction.



2. Expand an instruction and double-click the Logic routine to open.



3. Edit your logic using the available language editors.

Add-On Instructions Execution Considerations

An Add-On Instruction is executed just like any other routine belonging to a particular task. Because another task can preempt a task containing an Add-On Instruction being executed, it is not guaranteed that an Add-On Instruction will always be executed to its completion. You can use the User Interrupt Disable / Enable (UID/UIE) instructions to block a task switch if you want to be sure the Add-On Instruction executes to completion before switching to another task.

Optimizing Performance Considerations

The performance depends on the structuring, configuration, and the amount of code in an Add-On Instruction. You can pass large amounts of data through a structure using an InOut Parameter. The size of data referenced by an InOut Parameter does not impact scan time and there is no difference between passing a a user-defined type tag or an atomic tag since it is passed by reference.

When a rung condition is false, any calls to an Add-On Instruction are still processed even though the Logic routine is not executed. The scan time can be affected when many instances of an Add-On Instruction are executed false. Be sure to provide instructions in your documentation if an Add-On Instruction can be skipped when rung condition is false.

Defining the Operation of an Instruction in Different Scan Modes

To provide Add-On Instructions with the same flexibility as built-in instructions, optional Scan mode routines can be configured allowing you to fully define the behavior of the instruction. The Scan mode routines do not initially exist for an Add-On Instructions as they are optional and must be created by the user depending upon the requirements of the instruction.

Like all built-in instructions in the controller, Add-On Instructions support the following four controller Scan modes:

- True - Logic is scanned as the result of a true rung condition or the EnableIn Parameter is set True.
- False - Logic is scanned as the result of a false rung condition or the EnableIn Parameter is set False. Instructions in the controller may or may not have logic that executes only when that instruction is scanned false.
- Prescan - Occurs when the controller either powers up in Run mode or transitions from Program to Run. Instructions in the controller may or may not have logic that executes only when that instruction is executed in Prescan mode.
- Postscan - Occurs as a result of an Action in an SFC routine becoming inactive if SFCs are configured for Automatic Reset. Instructions in the controller may or may not have logic that executes only when that instruction is executed in Postscan mode.

The default behavior for executing an Add-On Instruction with no optional scan routines created may be sufficient for the intended operation of the instruction. If you do not define an optional Scan Mode the following default behavior of an Add-On Instruction occurs:

- True - Executes the main Logic routine of the Add-On Instruction.
- False - Does not execute any logic for the Add-On Instruction and does not write any outputs. Input Parameters are passed values from their arguments.
- Prescan - Executes the main Logic routine of the Add-On Instruction in Prescan mode. Any required input and output Parameters' values are passed.
- Postscan - Executes the main Logic routine of the Add-On Instruction in Postscan mode.

For each Scan mode, you can define a routine that is programmed specifically for that Scan mode and can be configured to execute in that mode.

- True - The main Logic routine for the Add-On Instruction executes (not optional).
- False - The EnableIn False routine executes normally in place of the main Logic when a scan false of the instruction occurs. Any required (or wired in FBD) input and output Parameters' values are passed.
- Prescan - The Prescan routine executes normally after a prescan execution of the main Logic routine. Any required input and output Parameters' values are passed.
- Postscan - The Postscan routine executes normally after a postscan execution of the main Logic routine.

Enabling Scan Modes

The Scan Modes tab in the Instruction Definition Editor lets you create and enable execution of the routines for the three Scan modes: Prescan, Postscan, and EnableInFalse.

Prescan Routine

When the controller transitions from Program mode to Run mode or when the controller powers up in Run mode, all logic within the controller is executed in Prescan mode. During this scan, each instruction may initialize itself and some initialize any tags it may reference. For most instructions, Prescan mode is synonymous with scanning false. For example, an OTE instruction clears its output bit when executed during Prescan. For others, special initialization may be done, such as an ONS instruction setting its storage bit during Prescan. During Prescan mode, all instructions evaluate false so conditional logic does not execute.

The optional Prescan routine for an Add-On Instruction provides a way for an Add-On Instruction to define additional behavior for Prescan mode. When a Prescan routine is defined and enabled, the Prescan routine executes normally after the primary Logic routine executes in Prescan mode. This is useful when it is desired to initialize tag values to some known or predefined state prior to execution. For example, setting a PID instruction to Manual mode with a 0% output prior to its first execution or to initialize some coefficient values in your Add-On Instruction.

TIP

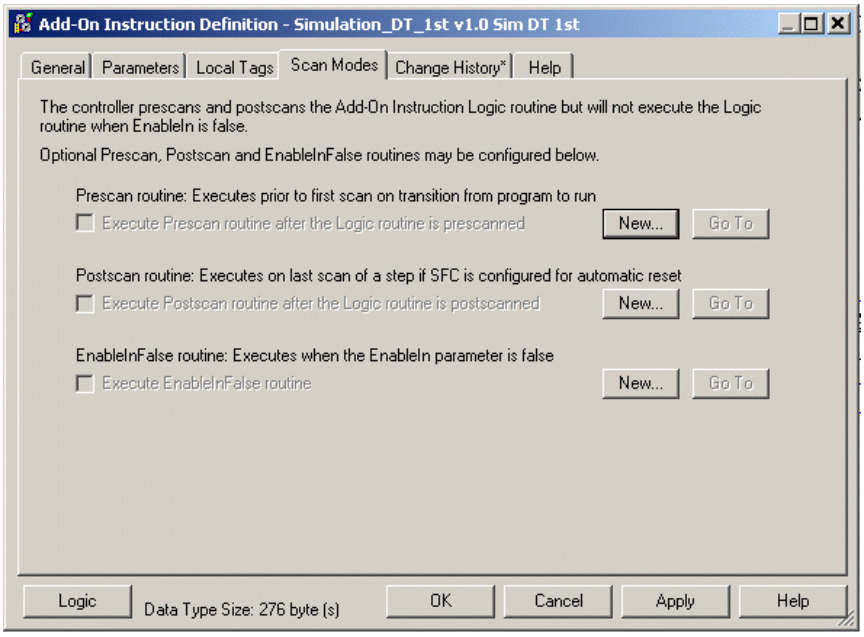
When an Add-On Instruction executes in Prescan mode, any required Parameters have their data passed.

- Values are passed to Input Parameters from their arguments in the instruction call.
- Values are passed out of Output Parameters to their arguments defined in the instruction call.

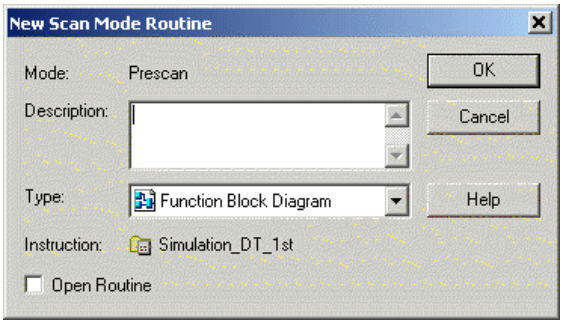
These values are passed even when the rung condition is false in Ladder Diagram or when the instruction call is in a false conditional statement in Structured Text. When Function Block Diagram routines execute, the data values are copied to all wired inputs and from all wired outputs, whether the Parameters are required or not.

Follow these steps to create a Prescan routine.

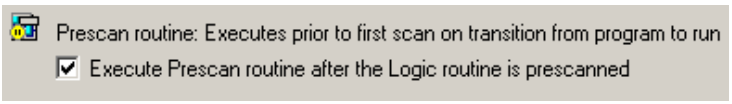
1. In the Controller Organizer, right-click an instruction and select Open Definition.
2. Click the Scan Modes tab.
3. Click New for Prescan Routine.



The New Scan Mode Routine dialog displays.



4. Select the type of programming language: Ladder Diagram, Function Block, or Structured Text.
5. Write a description of the Prescan behavior.
6. Click OK to create the routine and return to the Scan Modes tab.
7. Define if the prescan routine executes (or not) with the checkbox.



The Prescan routine can now be edited like any other routine.

Postscan Routine

Postscan mode only occurs for logic in an SFC Action when the Action becomes inactive and the SFC language is configured for Automatic Reset (which is not the default option for SFC). When an SFC Action becomes inactive, then the logic in the Action is executed one more time in Postscan mode. This mode is similar to Prescan in that most instructions simply execute as if they have a false condition. It is possible for an instruction to have different behavior during Postscan than it has during Prescan.

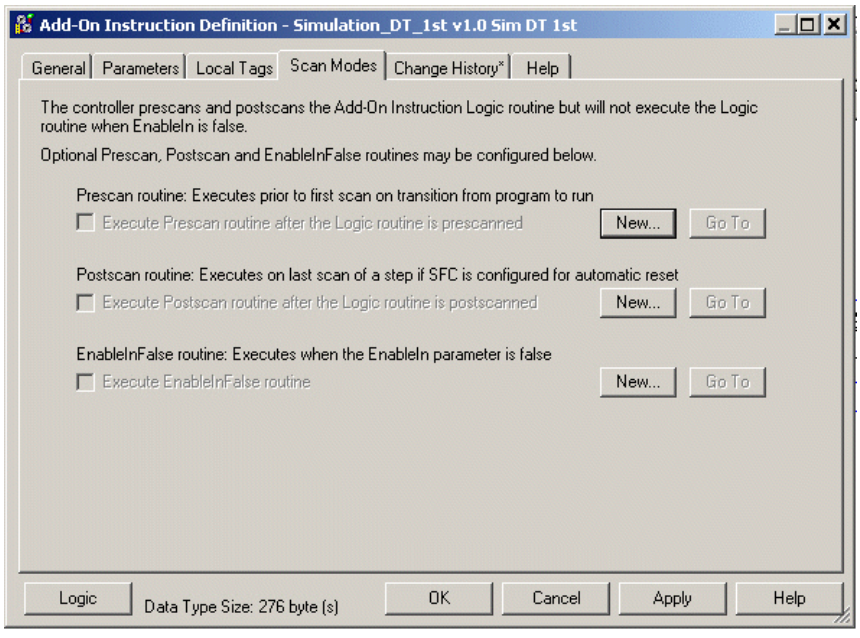
When an Add-On Instruction is called by logic in an SFC Action or a call resides in a routine called by a JSR from an SFC Action, and the Automatic Reset option is set, the Add-On Instruction executes in Postscan mode. The primary Logic routine of the Add-On Instruction executes in Postscan mode. Then if it is defined and enabled, the Postscan routine for the Add-On Instruction executes. This could be useful in resetting internal states, status values, or de-energizing instruction outputs automatically when the action is finished.

Follow these steps to create a postscan routine.

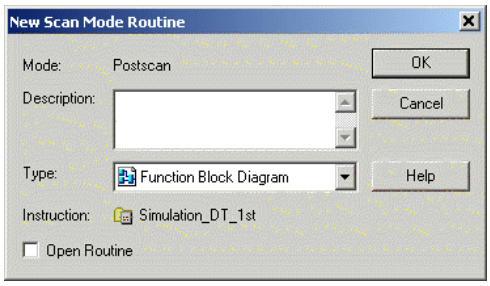
1. In the Controller Organizer, right-click an instruction and select Open Definition.

2. Click the Scan Modes tab.

3. Click New for Postscan Routine.



The New Scan Mode Routine dialog displays.

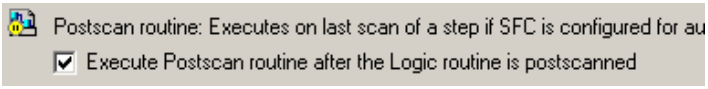


4. Select the type of programming language: Ladder Diagram, Function Block, or Structured Text.

5. Write a description of the postscan behavior.

6. Click OK to create the routine and return to the Scan Modes tab.

7. Define if the postscan routine executes (or not) with the checkbox.



The Postscan routine can now be edited like any other routine.

EnableInFalse Routine

When defined and enabled for an Add-On Instruction, the EnableInFalse routine executes when the rung condition is false or if the EnableIn Parameter of the Add-On Instruction is false (0). This is useful primarily for scan false logic, when used as an output instruction in a Ladder routine. A common use of scan false is the setting of OTE's to the de-energized state when the preceding rung conditions are false. An Add-On Instruction can use the EnableInFalse capability to let you to define behavior for the False conditions.

TIP

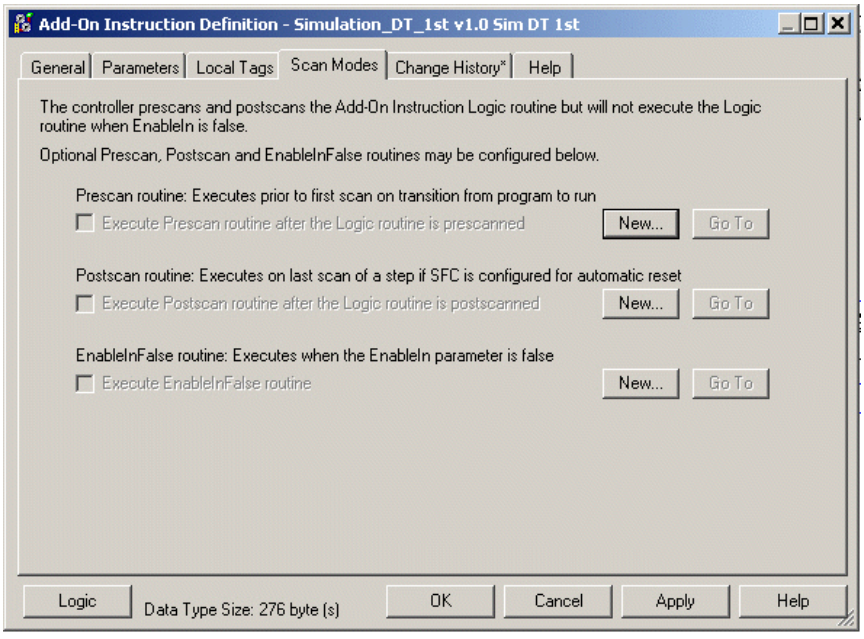
When the Add-On Instruction is executed in the false condition and has an EnableInFalse routine defined and enabled, any required Parameters have their data passed.

- Values are passed to Input Parameters from their arguments in the instruction call.
- Values are passed out of Output Parameters to their arguments defined in the instruction call.

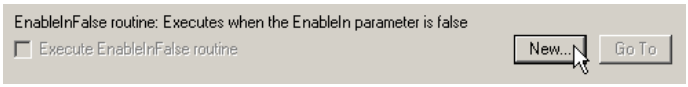
If the EnableInFalse routine is not enabled, the only action performed for the Add-On Instruction in the false condition is that the values are passed to any required Input Parameters in Ladder logic.

Follow these steps to create an EnableInFalse routine.

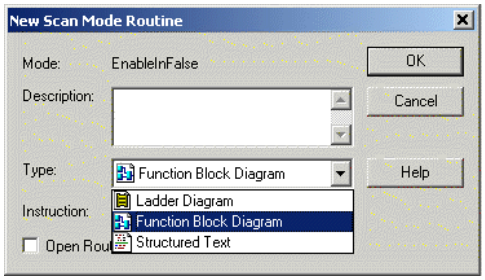
1. Right-click the instruction and select Open Definition.
2. Click the Scan Modes tab.



3. Click New on EnableInFalse routine.



The New Scan Mode Routine dialog displays.

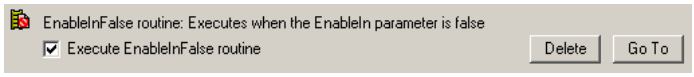


4. Select the Type pull-down list to select the language that you want to write the routine in Ladder, Function Block, or Structured Text.

5. Write a description of the EnableInFalse behavior.

6. Click OK to add an EnableInFalse routine to the Add-On Instruction definition.

7. Define if EnableIn False routine executes (or not) by checking Execute EnableInFalse routine.



The EnableInFalse routine can now be edited like any other routine.

Using the EnableIn and EnableOut Parameters

The EnableIn and EnableOut Parameters that appear by default in every Add-On Instruction have behaviors that conform to the three language environments: Ladder Diagram, Function Block Diagram, and Structured Text.

In order to execute the primary Logic routine in any of the language environments, the EnableIn Parameter must be True (1). In general, the EnableIn Parameter should not be referenced by the primary Logic routine within the instruction definition. The EnableOut parameter will, by default, follow the state of the EnableIn parameter but can be overridden by user logic to force the state of this Parameter.

TIP If EnableIn is false, then EnableOut cannot be made true in an EnableIn False routine.

If the EnableIn parameter of the instruction is False (0), the Logic routine is not executed and the EnableOut parameter is set False (0). If an EnableInFalse routine is included in the instruction definition and it is enabled, the EnableInFalse routine will be executed.

EnableIn Parameter and Ladder Diagram

In the Ladder diagram environment, the EnableIn parameter reflects the rung state on entry to the instruction. If the rung state preceding the instruction is True (1), the EnableIn parameter will be True and the primary Logic routine of the instruction will be executed. Likewise, if the rung state preceding the instruction is False (0), the EnableIn parameter will be False and the primary Logic routine will not be executed.

TIP An instruction with only input Parameters, except EnableOut, is treated as an input instruction in a Ladder diagram. The EnableOut parameter is used for the rung condition.

EnableIn Parameter and Function Blocks

In the Function Block environment, the EnableIn Parameter can be manipulated by the user through its pin connection. If no connection is made, the EnableIn Parameter is set True (1) when the instruction begins to execute and the primary Logic routine of the instruction will be executed. If a wired connection to this Parameter is False (0), the primary Logic routine of the instruction will not execute. Another reference writing to the EnableIn Parameter, such as a Ladder Diagram rung or a Structured Text assignment, will have no influence on the state of this Parameter. Only a wired connection to this Parameter's input pin can force it to be False (0).

EnableIn Parameter and Structured Text

In the Structured Text environment, the EnableIn Parameter is always set True (1) by default. The user cannot influence the state of the EnableIn Parameter in a Structured Text call to the instruction. Since EnableIn is always True (1) in Structured Text, the EnableInFalse routine will never execute for an instruction call in Structured Text.

Testing the Add-On Instruction

You need to test and troubleshoot the logic of an instruction to get it working.

TIP When a fault occurs in an Add-On Instruction routine, a fault log is created that contains extended information useful for troubleshooting.

Before You Test

Before you start to test an Add-On Instruction, do the following.

1. Open a project to debug offline.

   TIP Add-On Instruction definitions can only be created or modified when offline. You can add, delete, or modify tag arguments using online editing.

2. Add the Add-On Instruction to the project, if it is not already there.

Test the Flow

1. Add a call to the instruction in a routine in the open project.
2. Assign any arguments to required Parameters for your call.
3. Download the project.

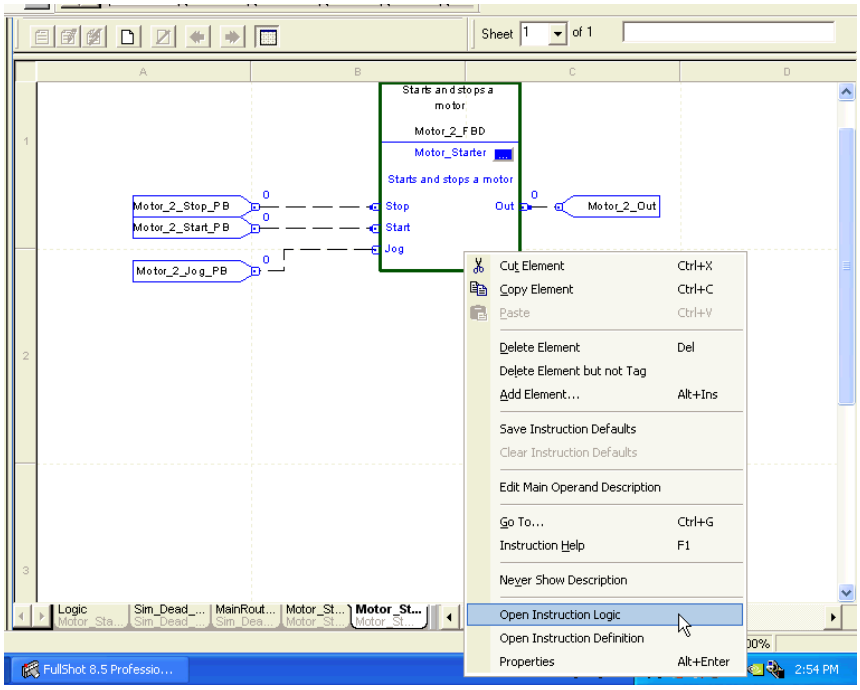
Monitor Logic with Data Context Views

You can simplify the online monitoring and troubleshooting of your Add-On Instruction by using Data Context views. The Data Context selector lets you select a specific call to the Add-On Instruction that defines the calling instance and arguments whose values are used to visualize the logic for the Add-On Instruction.

TIP When troubleshooting an Add-On Instruction, use a non-arrayed instance tag for the call to the instruction. This lets you monitor and troubleshoot the instruction's Logic routine with a data context. Variable indexed arrays cannot be used to monitor the logic inside an Add-On Instruction.

1. Go into Run mode.

2. Right-click the instruction call and select Open Instruction Logic.

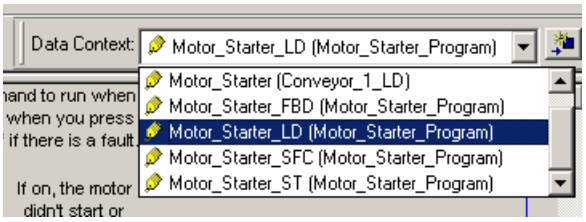


The Logic routine opens with animated logic for the specific calling instance.

3. Monitor the Logic routine with data context.

TIP

You can also show the Add-On Instruction logic animated with other calling instances by using the Data Context menu located on the upper right of the logic dialog.



You can also monitor any defined Scan mode routines this way by opening the routine and selecting the calling instance in the Data Context.

Tips for Validating the Add-On Instruction in Different Scan Modes

The most straightforward method to verify Scan mode operation is to execute the instruction first with the Scan mode routine disabled, then again with it

enabled. Then you can determine whether the Scan mode routine performed as expected.

Verifying Individual Scan Modes

Instruction	Description
True	This is simply the execution of the main Logic routine.
False	In a Ladder Logic target routine, this entails placing an XIC before an instance of the instruction and evaluating instruction results when the XIC is false. In a Function Block target routine, this entails executing an instance of the instruction with the EnableIn Parameter set to zero(0).
Prescan	Place the controller in Program mode, then place it in Run mode.
Postscan	With the controller configured for SFC Automatic Reset, place an instance of the instruction into the Action of an SFC. Run the SFC such that this Action is executed and the SFC proceeds beyond the step that is associated with this Action.

Managing Add-On Instruction Definitions

This section describes the different ways you can manage your Add-On Instructions.

Storing Your Instructions

There are two ways to store a group of instruction definitions together. The first way is to store your Add-On Instructions in a project file.

1. Identify what instructions you want to store.
2. Place them in a project file called something like MyInstructions.ACD.
3. Open other projects in additional instances of RSLogix5000 software and use copy and paste or drag and drop to move a copy of the instruction definitions from one project to another.

   If any of these instruction definitions reference the same Add-On Instruction or User-Defined Type, there is only one shared copy in the project file. When an Add-On Instruction is copied to another project, it also copies any definition it references to the target project.

The second way is to store your Add-On Instruction definition in an .L5X export file.

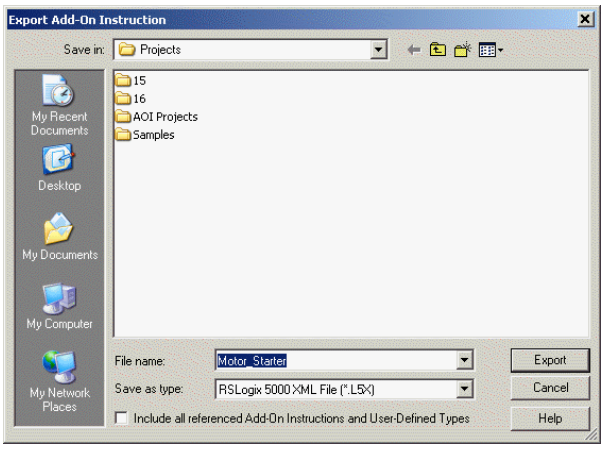
When you select to Export an Add-On Instruction, that Add-On Instruction definition is saved to the export file. Optionally, you can include any nested Add-On Instruction definitions or User-Defined Data Types that are referenced by the exported Add-On Instruction.

In deciding how to manage your Add-On Instruction definitions in export files, you need to consider your goals in storing the definitions.

Do you want to store many Add-On Instructions that share a set of common Add-On Instructions or User-Defined Types in a common location?

if so, you may want to choose to export each Add-On Instruction and User-Defined Types to separate files without including references by using these steps.

1. Select an Add-On Instruction in the Controller Organizer.
2. Right-click the instruction and select Export Add-On Instruction.
3. Choose the common location to store the L5X file.
4. Type a name for the file.



5. Uncheck the Include referenced Add-On Instructions and User-Defined Types checkbox.

6. Click Export.

7. Follow the above steps to individually export the other shared Add-On Instructions and User-Defined Types.

Using export in this way lets you manage the shared Add-On Instruction and User-Defined Type definitions independently of the definitions that reference them. One advantage of this is the ability to update the shared component without having to regenerate all the export files for the definitions that reference it. That is, it is only stored in one file instead of in every file whose definition references it. This helps with the maintenance of the definitions as you only have to update one export file.

When Add-On Instructions have been exported this way, to use them in a project you must import any definition that it references before the referencing one can import successfully. To do this, work bottom up by importing nested Add-On Instructions or referenced User-Defined Types, and then the Add-On Instructions that reference them.

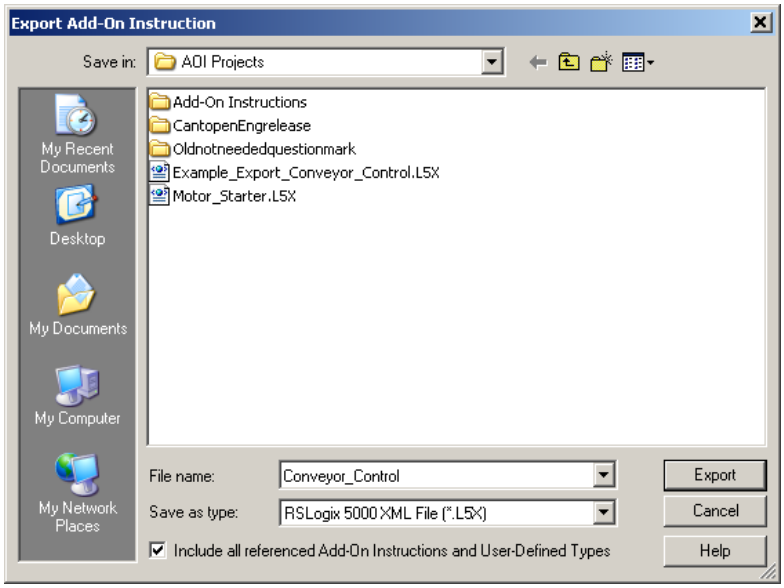
Do you want to distribute an Add-On Instruction definition as one file or to manage each Add-On Instruction as a stand-alone definition?

If so, you may want to export the instruction definition and any referenced Add-On Instructions or User-Defined Types into one export file by using these steps.

1. Right-click Add-On Instruction in Controller Organizer and select Export Add-On Instruction.

2. Choose the location to store the L5X file.

3. Type a name for the file.



4. Check the Include referenced Add-On Instructions and User-Defined Types checkbox.

5. Click Export.

This exports the selected Add-On Instruction definition and all the referenced definitions into the same export file. This file can be used to distribute an Add-On Instruction definition. When the exported Add-On Instruction is imported into the project, the referenced definitions are imported as well in one step.

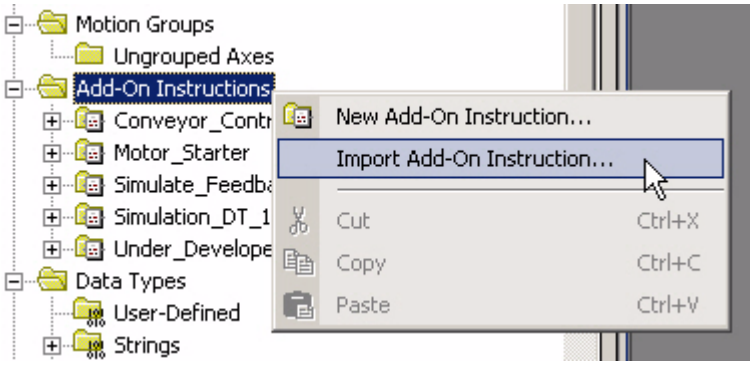
Updating an Add-On Instruction to a Newer Revision

When you need to update an instruction, you must be working offline. If you need a newer revision of an Add-On Instruction, import from an L5X file or copy it from an existing project.

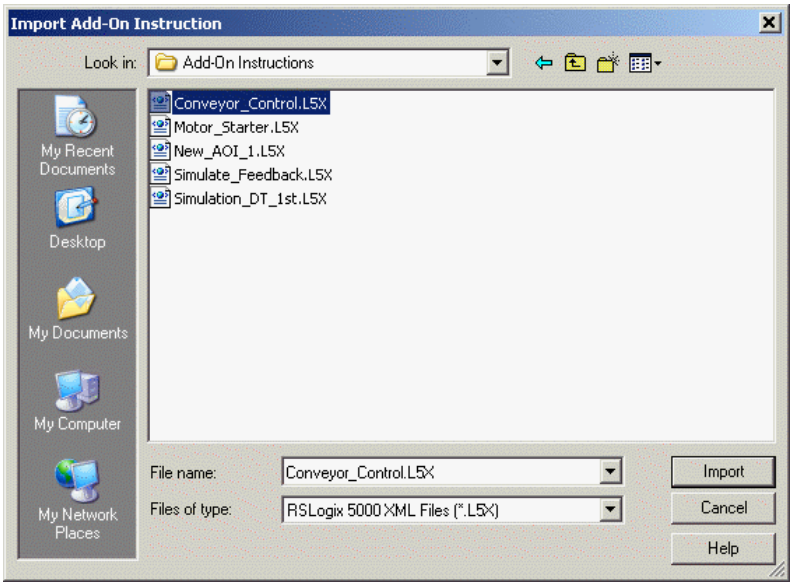
Follow these steps to update an Add-On Instruction to a newer revision.

IMPORTANT Be careful when updating an Add-On Instruction. You need to ensure that your existing code will work correctly with the new version of the instruction.

1. Right-click the Add-On Instruction folder and choose Import Add-On Instruction.



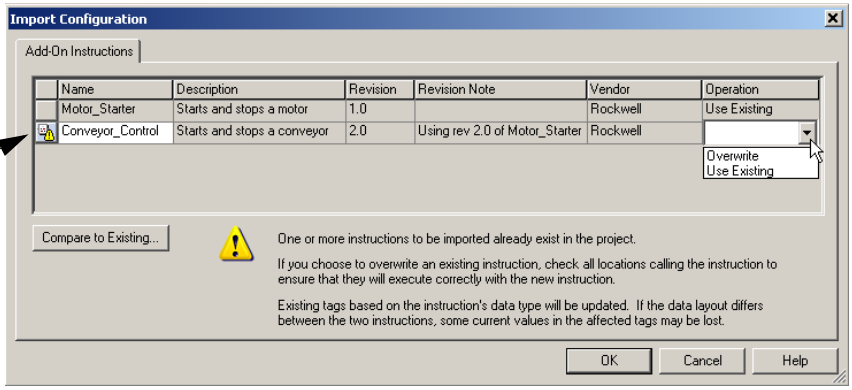
2. Select the file with the Add-On Instruction definition and click Import.



3. Review the Import Configuration dialog.

   It shows two definitions because the Conveyor_Control references Motor_Starter and both definitions are in the import file.

The icon  in the row header indicates that the instruction definition collides with one in the project.

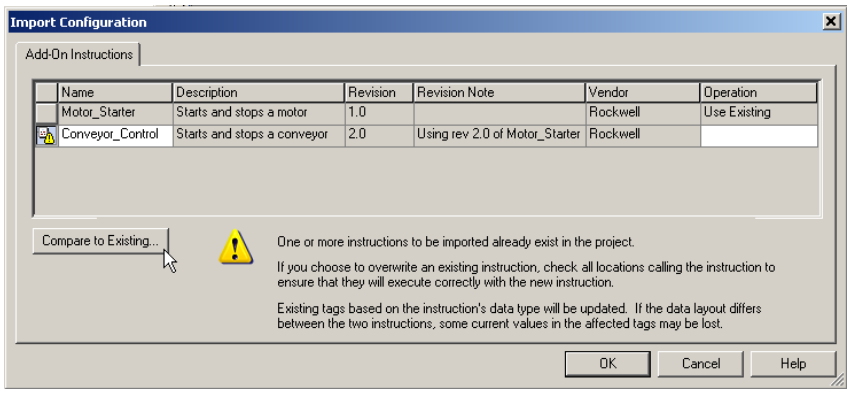


   For any instruction definition being imported that collides with one in the project, there are two options for the Add-On Instructions, Overwrite or Use Existing.

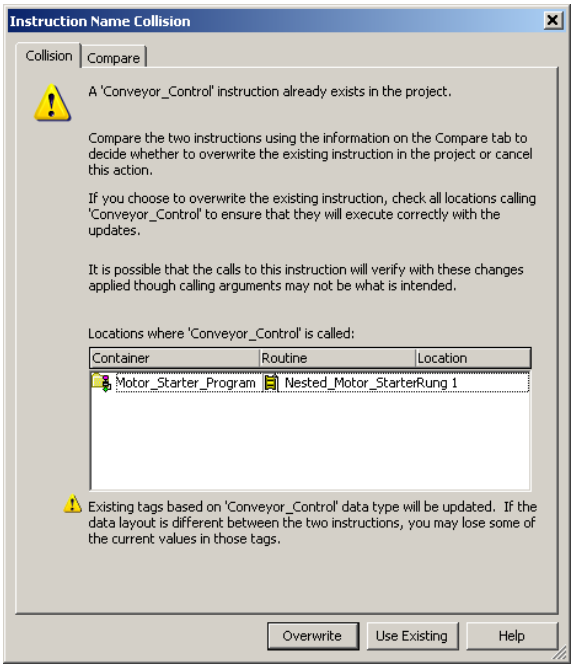
4. Select a row and click the Compare to Existing button to see the Collision dialog.

   Use the Collision dialog to see the differences and where the Add-On Instruction is used.

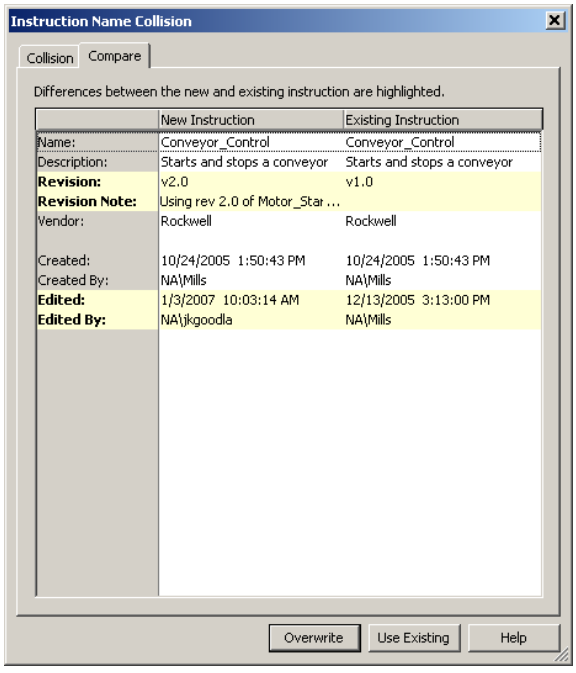
5. Click on Compare to Existing to open the Collision dialog.



The Collision tab shows where the existing Add-On Instruction is used.



The Compare tab shows what is different between the definitions, in this case, the revision information.



TIP The Compare dialog only compares metadata for each instruction definition, such as description, revision or edited date. For effective revision control, enter a detailed revision note.

6. Select Use Existing, to leave the definition in the project as is, or Select Overwrite to replace the Add-On Instruction in your project with the definition that you are importing.

7. Click OK on the Import Configuration dialog to complete the operation.

Defining Source Protection for an Add-On Instruction

You can apply source protection to protect your intellectual property or prevent unintended edits of validated source. Source protecting your Add-On Instructions protects your intellectual property and avoids unwanted changes.

Applying Source Protection

You can use the Source Protection capability to limit a user of your Add-On Instruction to read-only access or to allow no access to the internal logic or Local Tags used by the instruction.

You can protect their use and modification with a source key file when you distribute them. This lets you stop unwanted changes to your instruction as well as protects your intellectual property.

Refer to Setting Up Source Protection on page 428 for more information.

Sometimes you will want to protect the source definition of an Add-On Instruction from the view of a user. This may be due to the proprietary nature of the algorithms or for strict revision control.

- Source Protected

  Users without the source key cannot view any routine or Local Tags, or make any edits to the Add-On Instruction.

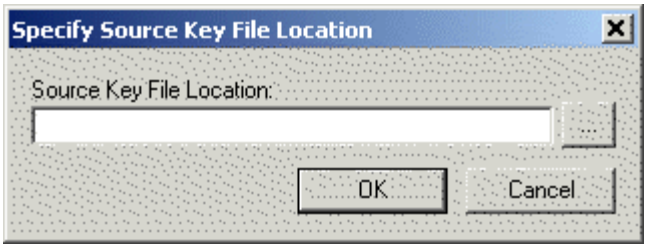
- Source Protected with Viewable Option

  Users without the source key can view all components of the Add-On Instruction including its logic and Local Tags, but are prevented from making any edits to the instruction.

Enable the Source Protection Feature

You may find that Source Protection is unavailable and not listed in your menus. To enable the Source Protection feature, enable it during the RSLogix 5000 System Install or use the tool RS5KSrcPtc.exe on your installation CD.

If it is the first time the Source Protection has been configured, a dialog displays asking to configure a source key file location.



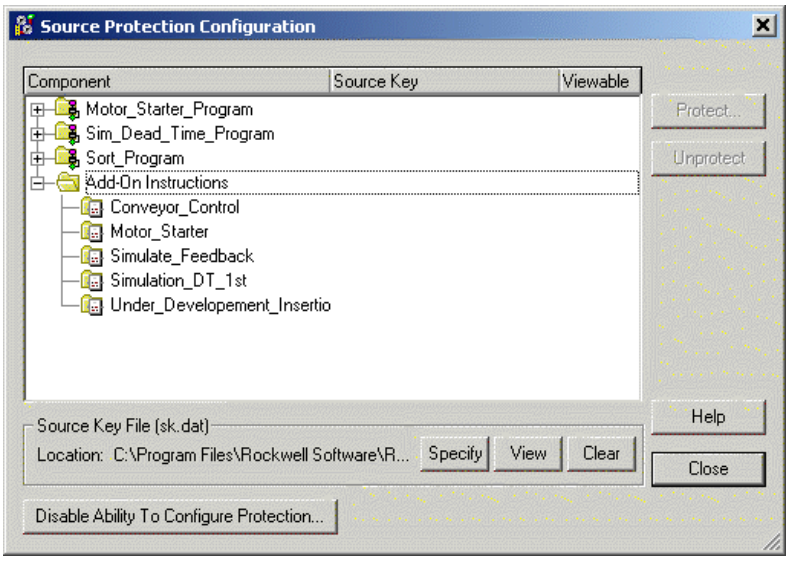
1. Enter the location of (or browse to) where the source key is kept, it may or may not exist yet.

2. Click OK.

Apply Source Protection

To source protect your Add-On Instruction follow these steps.

1. In RSLogix 5000 select Tools -> Security -> Configure Source Protection, to access the source configuration dialog.

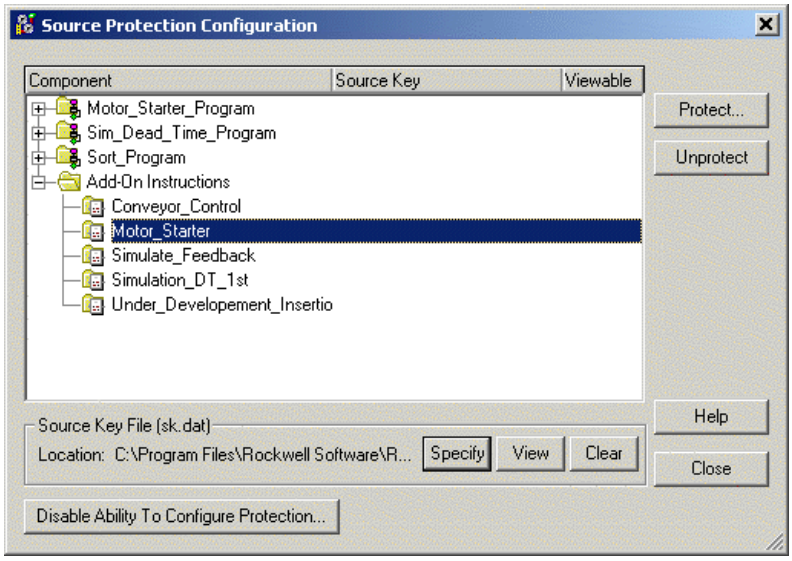


2. Click Specify to identify the location for the sk.dat Source Key File.

3. Click OK in both dialogs to accept.

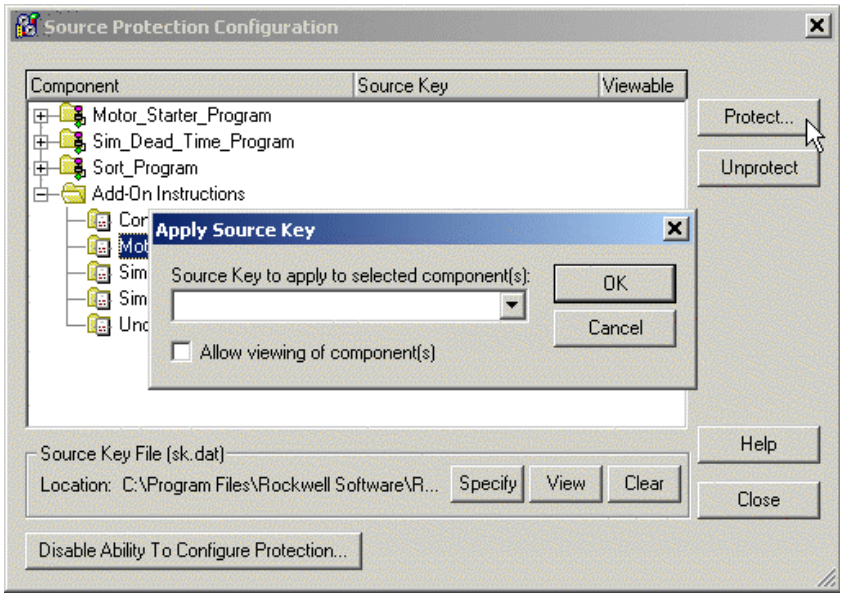
4. Click Yes to confirm creation of the sk.dat Key File.

5. Select the component you want to protect..



6. Click Protect.

The Source Key Entry dialog displays. This is where you enter the individual source key for this component. These source keys follow the conventions for routine source keys.



7. Expand the Add-On Instructions folder to view the instructions available to apply source protection.

   The components include all Routines and Add-On Instruction definitions in the project.

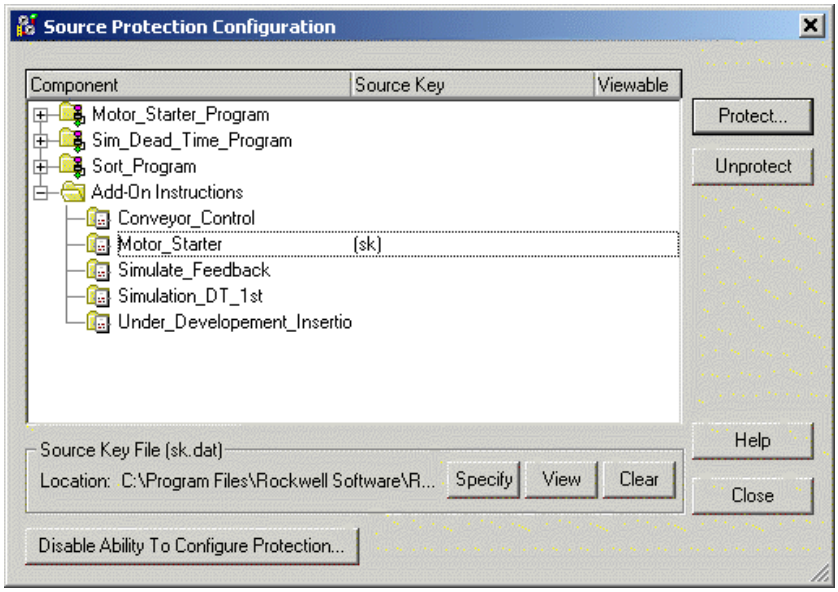
8. Select the Add-On Instruction you wish to protect and click Protect.

9. Enter a name for the source key for this component or use an existing one.

10. If you wish to allow users to be able to see the logic and Local Tags, check the box Allow viewing of component(s).

    The Source Protection Configuration should now resemble the following dialog.

IMPORTANT Before you click Close, you must first either click Clear or Disable Ability to Configure Source Protection or remove the sk.dat file from the PC so that the source key is no longer present, if you wish to observe the source protection settings.



11. Click Close.

12. Save the project.

13. Close RSLogix 5000 software.

Observe Source Protection

If you want to observe how source protection works, do the following steps:

1. Rename the sk.dat file to some other name.

2. Start RSLogix 5000 software and open the project.

   There is now no valid key file available to this RSLogix 5000 project.

   TIP When the Source Key is available, the Add-On Instruction will behave the same as if it were not source protected.

3. Review how the instruction appears in the Add-On Instructions folder.

   Now the protected Add-On Instruction definition's routines and tag folder are not shown if fully protected, and the definition cannot be edited.

Creating Instruction Help

Custom instruction help is generated automatically as you are creating your Add-On Instructions and as you add information in the description fields. You can also provide additional help documentation by entering it on the Help tab of the Add-On Instruction Definition Editor. The instruction help is available in the instruction browser and from any call to the instruction in a language editor by pressing F1.

Custom instruction help makes it easier for users to get the help they need for using the instructions.

Define the Documentation for Add-On Instructions

RSLogix 5000 software automatically builds help for your Add-On Instructions using the instruction's description, revision note, and Parameter descriptions. By creating meaningful descriptions, you can help the users of your instruction. In addition, you can add your own custom text to the help by using the extended description field.

Write Clear Descriptions

When writing your descriptions keep the following in mind.

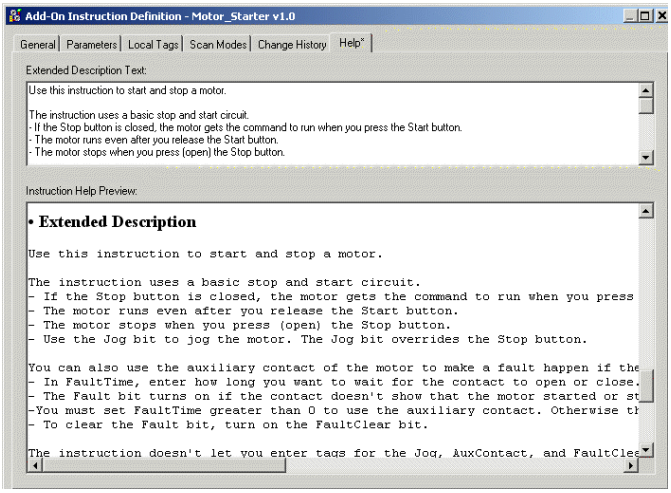
- Use short sentences and simple language
- Be brief and direct when you write
- Offer simple examples
- Proof read your field entries.

Extended Description Text

This is an example of the Extended Description Text field in the Help tab of the Instruction Definitions Editor. This area lets you create directions on how to use your instruction and troubleshooting information. The Instruction Help Preview window shows how your text will look as generated instruction help.

TIP

When you are entering your text into the Extended Description Text field you can use returns and tabs in the field to format the text, and if you copy and paste text into the field tabs are preserved.



Document an Add-On Instruction

Follow these steps to create custom help for an instruction.

1. Right-click an Add-On Instruction and select Open Definition.

2. On the General tab, enter a description and a revision note for the Add-On Instruction to explain the purpose of the instruction.

3. Click the Parameters tab and enter a meaningful description for each Parameter.

4. Right-click each routine located below the Add-On Instruction in the Controller Organizer and select Properties.

5. Enter description for execution of each routine:
   a. For the Logic routine, describe execution of the instruction when EnableIn is true.
   b. For the EnableInFalse routine (if one exists), describe actions that will take place when EnableIn is false, such as any outputs that get cleared.
   c. For the Prescan routine (if one exists), briefly describe actions that will take place during Prescan, such as initialization of any Parameters.
   d. For the Postscan routine (if one exists), briefly describe actions that will take place during Postscan, such as initialization of any Parameters resetting any internal state of the instruction.

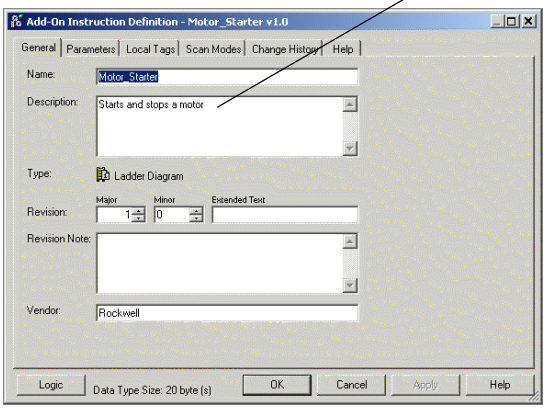
6. Click the Help tab of the Add-On Instruction definition and enter additional information in the Extended Description field.

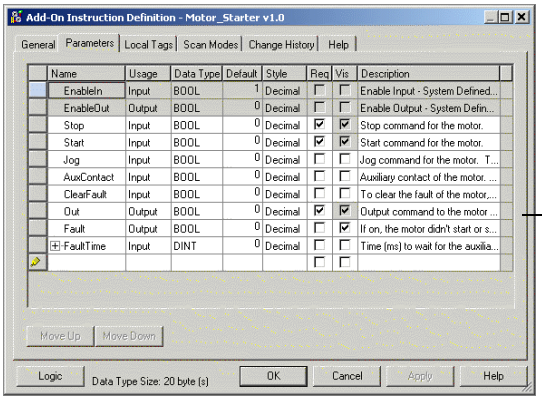
   The extended description can include the following information.

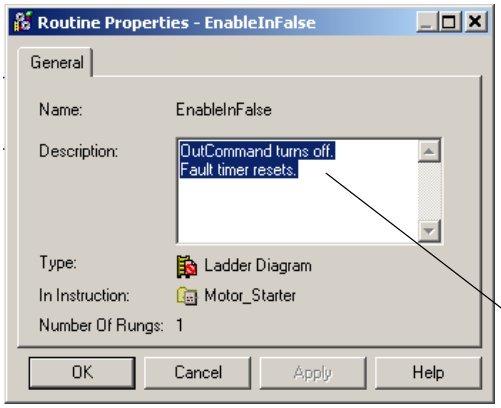
   - Additional Parameter information
   - Description of how the instruction executes
   - Change history notes

7. Review the Help format in the preview window.

This is an example of the RSLogix 5000 software generated help for the instruction. This information is gathered from the definition descriptions that you complete when defining an instruction.







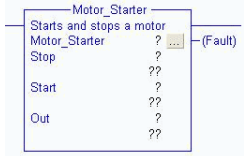
Motor_Starter v1.0

Rockwell
[Contact the Add-On Instruction developer for questions or problems with this instruction]

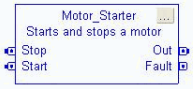
Starts and stops a motor

• Available Languages

Relay Ladder



Function Block



Structured Text

```
Motor_Starter(Motor_Starter,Stop,Start,Out);
```

• Parameters

Required	Name	Data Type	Usage	Description
x	Motor_Starter	Motor_Starter	InOut	
	EnableIn	BOOL	Input	
	EnableOut	BOOL	Output	
x	Stop	BOOL	Input	Stop command for the motor.
x	Start	BOOL	Input	Start command for the motor.
	Jog	BOOL	Input	Jog command for the motor. To jog the motor, turn on this bit. To stop the jog, turn off this bit.
	AuxContact	BOOL	Input	Auxiliary contact of the motor. Make sure you set the FaultTime. Otherwise, this input doesn't do anything.
	ClearFault	BOOL	Input	To clear the fault of the motor, turn on this bit.
x	Out	BOOL	Output	Output command to the motor starter. If on, the motor starts. If off, the motor stops.
	Fault	BOOL	Output	If on, the motor didn't start or stop.
	FaultTime	DINT	Input	Time (ms) to wait for the auxiliary contact to open or close. The Fault bit turns on when that time is up.

• Extended Description

```
Use this instruction to start and stop a motor. The instruction uses a basic stop and start circuit.
- If the Stop button is closed, the motor gets the command to run when you press the Start button.
- The motor runs even after you release the Start button.
- The motor stops when you press (open) the Stop button.
- Use the Jog bit to jog the motor. The Jog bit overrides the Stop button.

You can also use the auxiliary contact of the motor to make a fault happen if the motor doesn't start
- In FaultTime, enter how long you want to wait for the contact to open or close. Enter the time in mi
- The Fault bit turns on if the contact doesn't show that the motor started or stopped within the Faul
-You must set FaultTime greater than 0 to use the auxiliary contact. Otherwise the instruction doesn't
- To clear the Fault bit, turn on the FaultClear bit.

The instruction doesn't let you enter tags for the Jog, AuxContact, and FaultClear bits in the LD and
- In LD, use XIC and OTE instructions to read the value of the auxiliary contact tag and write it to t
- In ST, use an assignment (:=) to set the AuxContact bit equal to the value of the auxiliary contact
```

• Execution

Condition	Description
EnableIn is true	OutCommand turns on when Stop and Start are on. OutCommand turns off when Stop turns off.
EnableIn is false	OutCommand turns off. Fault timer resets.

• Revision v1.0 Notes

Motor Starter Instruction Example

The Motor_Starter Add-On Instruction starts and stops a motor.

If the stop pushbutton = off and the start pushbutton = on then

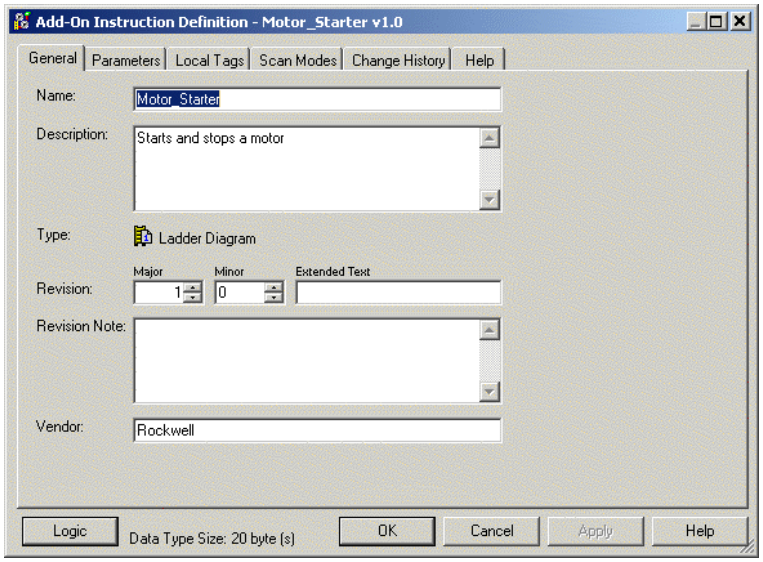
The conveyor gets the command to run.

The instruction seals in the command, so the conveyor keeps running even after you release the start pushbutton.

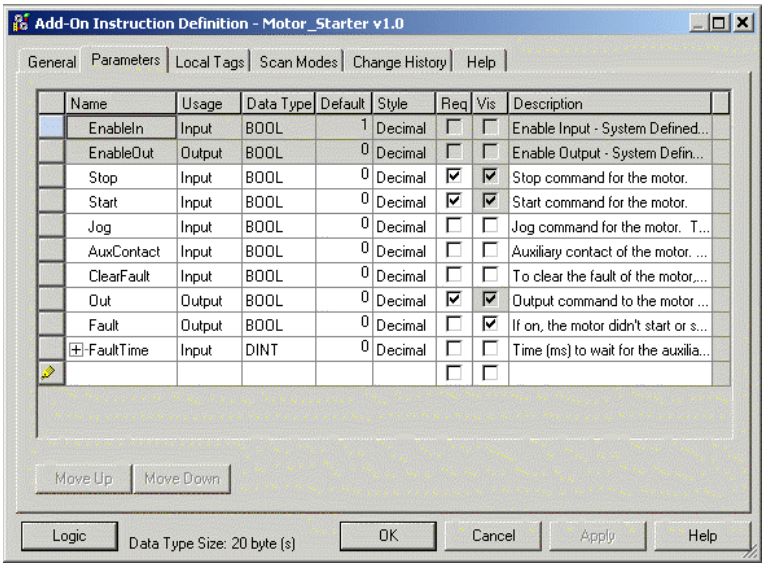
If the stop pushbutton = on then the conveyor stops.

The following three diagrams show the Motor Starter instruction called in different programming languages.

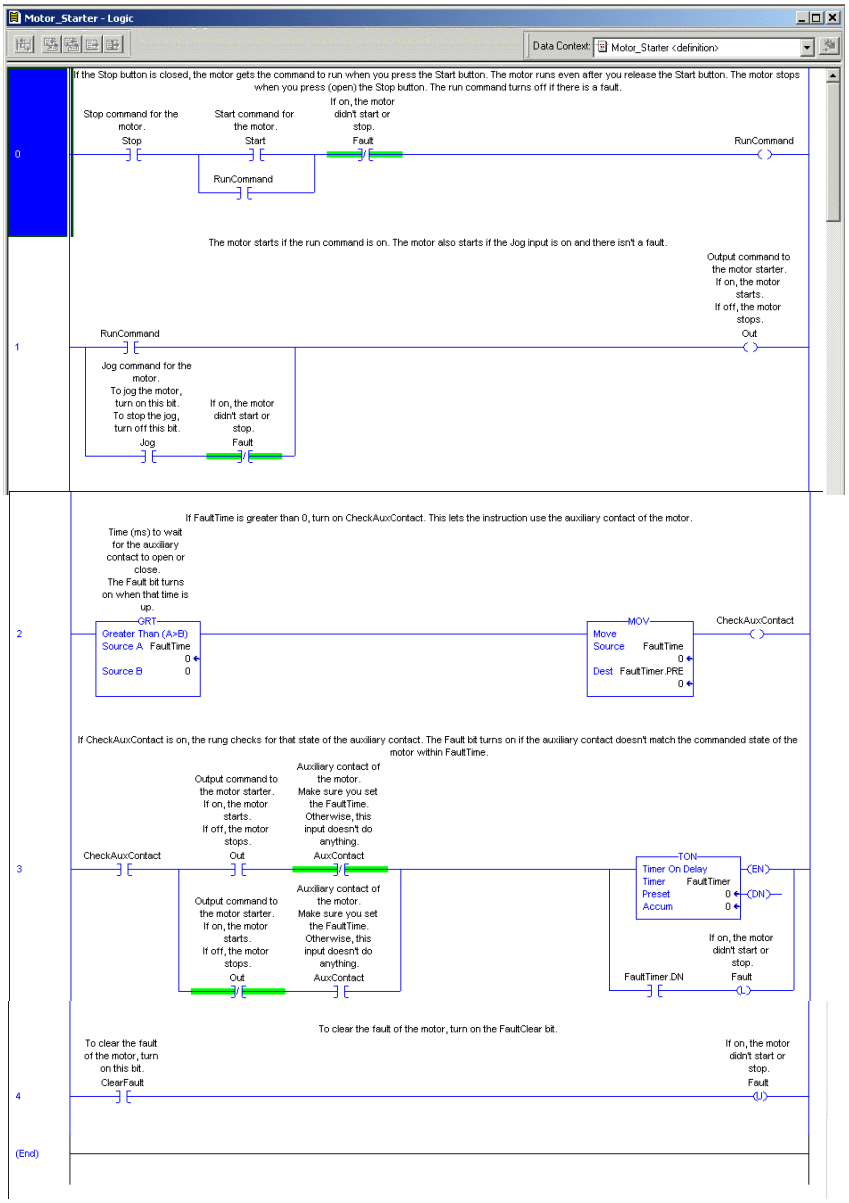
Motor Starter Example Definition Editor General Tab



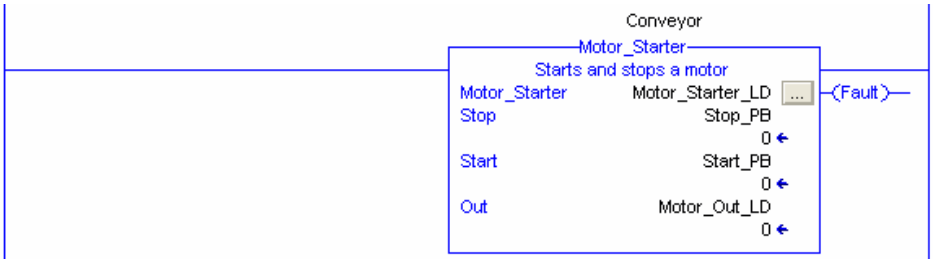
Motor Starter Example Definition Editor Parameter Tab



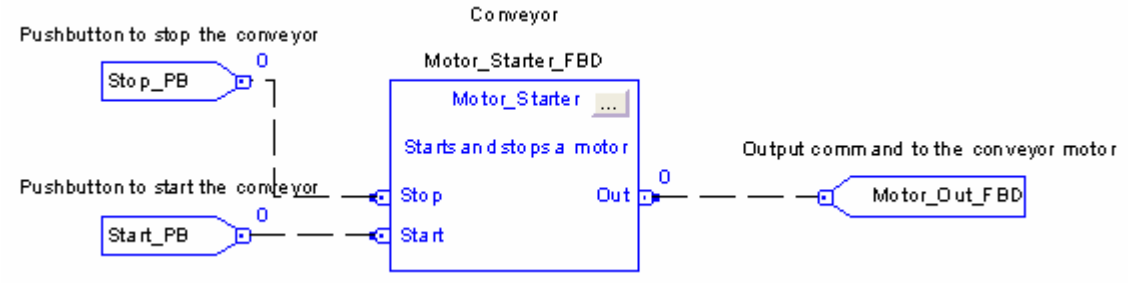
Motor Starter Example Ladder Logic



Motor Starter Ladder Diagram



Motor Starter Function Block Diagram



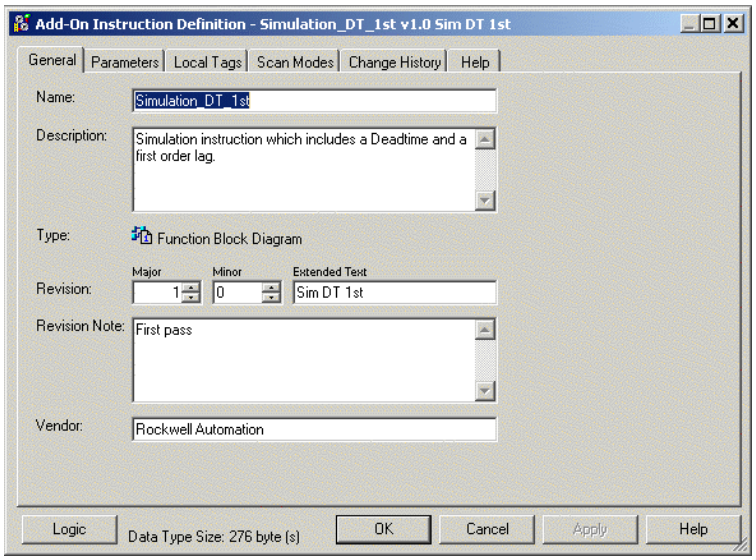
Motor Starter Structured Text

```
Motor_Starter(Motor_Starter_ST, Stop_PB, Start_PB,
Motor_Out_ST);
```

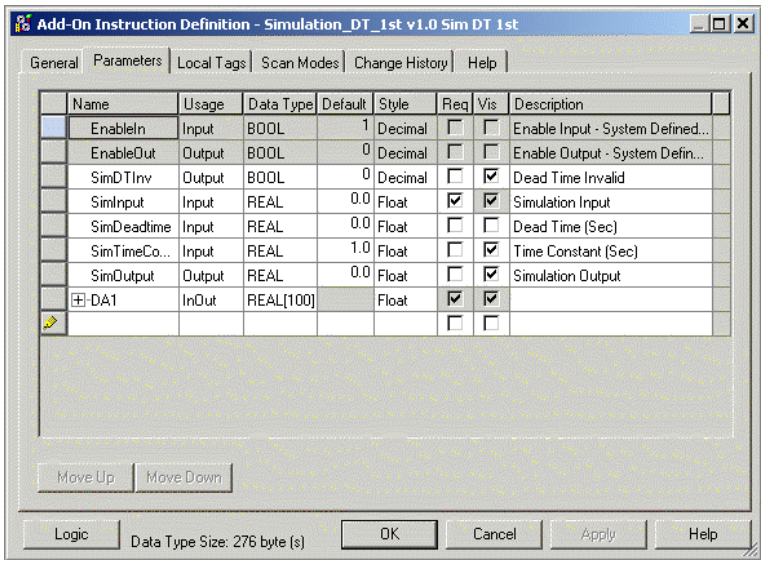
Simulation Instruction Example

The Simulation_DT_1st Add-On Instruction adds a dead time and a first-order lag to an input variable.

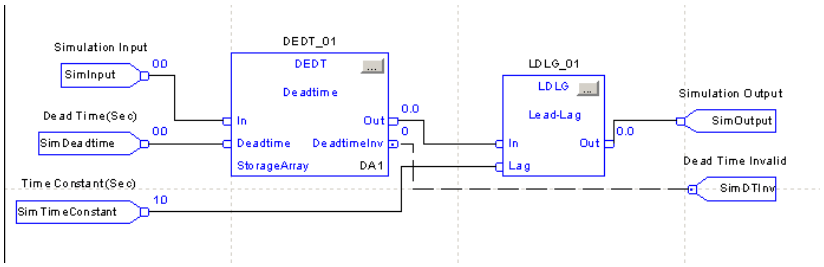
Simulation Example Definition Editor General Tab



Simulation Example Definition Editor Parameter Tab



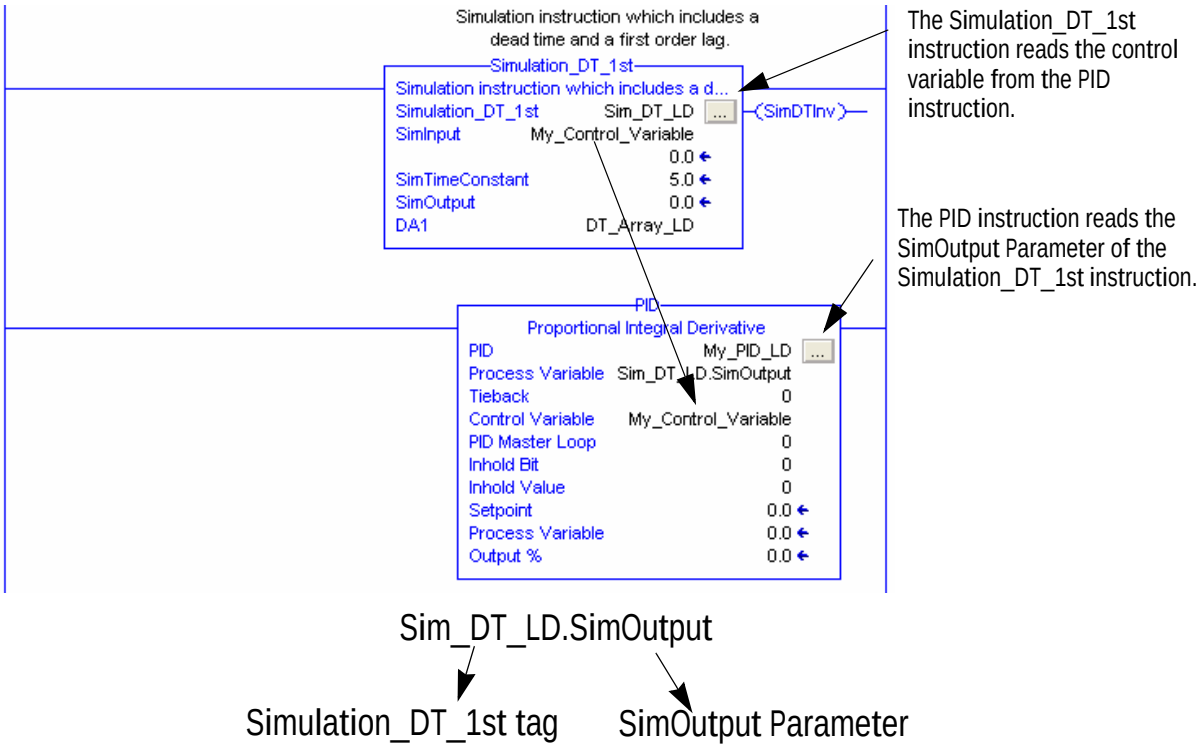
Simulation Example Logic



In this example, the instruction simulates a closed-loop control.

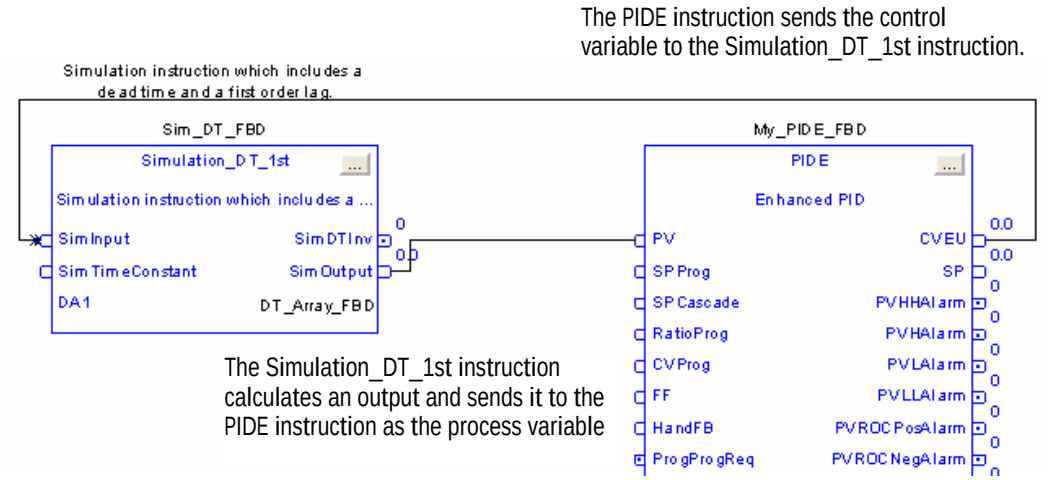
Ladder Diagram Configuration

The Simulation_DT_1st instruction reads the control variable from the PID instruction. The PID instruction reads the SimOutput Parameter of the Simulation_DT_1st instruction.

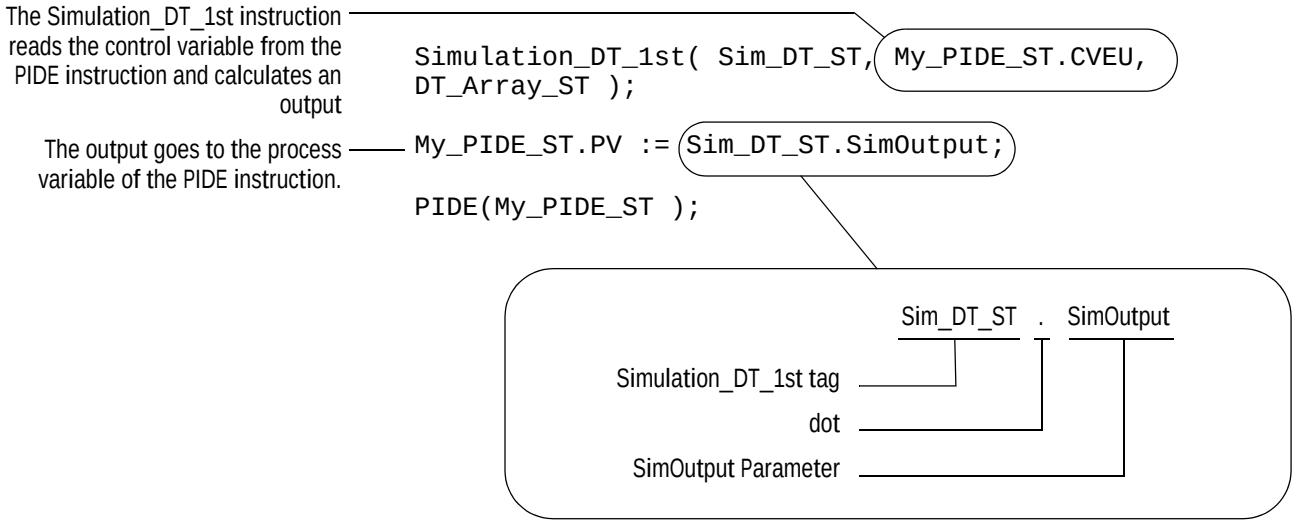


Function Block Diagram Configuration

The PIDE instruction sends the control variable to the Simulation_DT_1st instruction. The Simulation_DT_1st instruction calculates an output and sends it to the PIDE instruction as the process variable



Structured Text Configuration



Using an Add-On Instruction

Introduction

This chapter describes how to use an Add-On Instruction. Every dialog field is not explained in detail. Refer to the RSLogix 5000 Online Help to get detailed information.

Topic	Page
What is an Add-On Instruction?	313
Adding an Add-On Instruction to Your Routine	314
Programming Tips when Using an Add-On Instruction	319
Monitoring the Value of a Parameter	323
View Logic and Monitor with Data Context	324
Copying an Add-On Instruction Definition	327
Importing an Add-On Instruction Definition	328
Exporting an Add-On Instruction Definition	329

What is an Add-On Instruction?

Add-On Instructions are custom instructions that are added and available for use in a RSLogix 5000 project. They are similar to built-in instructions or subroutines that can be called by programming but are an entity to themselves so they become like any other RSLogix 5000 built-in instruction.

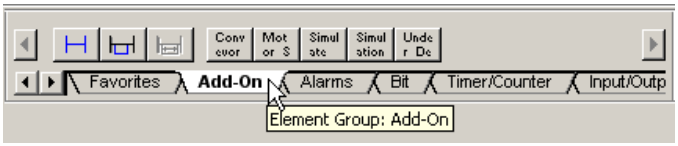
Adding an Add-On Instruction to Your Routine

Add-On Instructions are used in your routines like any built-in instructions. You add calls to your instruction and then define the arguments for any Parameters.

The Add-On Instruction can be used in any one of the Ladder Diagram, Function Block or Structured Text languages (including Structured Text within Sequential Function Chart actions). The appearance of the instruction will conform to the language in which it is placed.

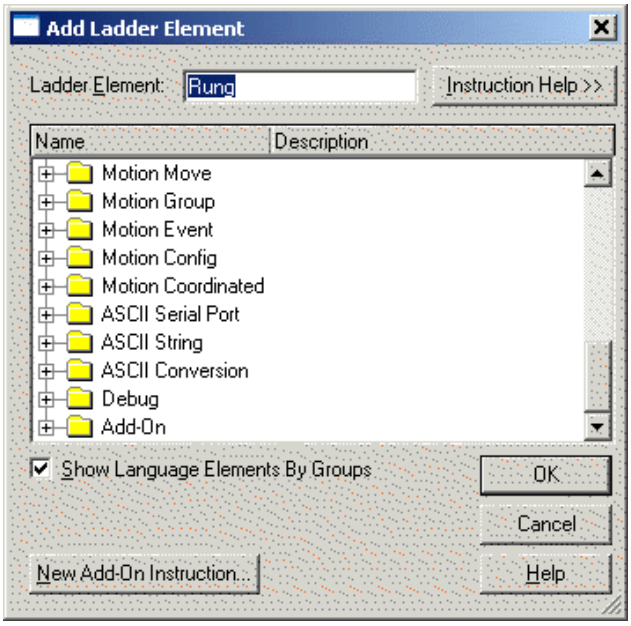
The Add-On Instructions in the project can be accessed from any of the normal instruction selection tools.

The instruction toolbar has an Add-On tab that lists all of the currently available Add-On Instructions in the project.

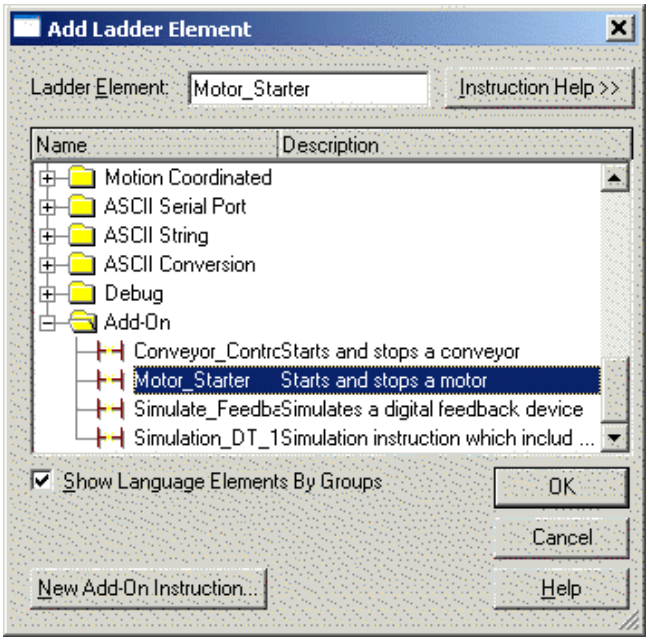


Follow these instructions to access the Add (language) Element dialog.

1. Press Alt + Insert keys anywhere in the editor or by right-clicking on the logic in the Editor.

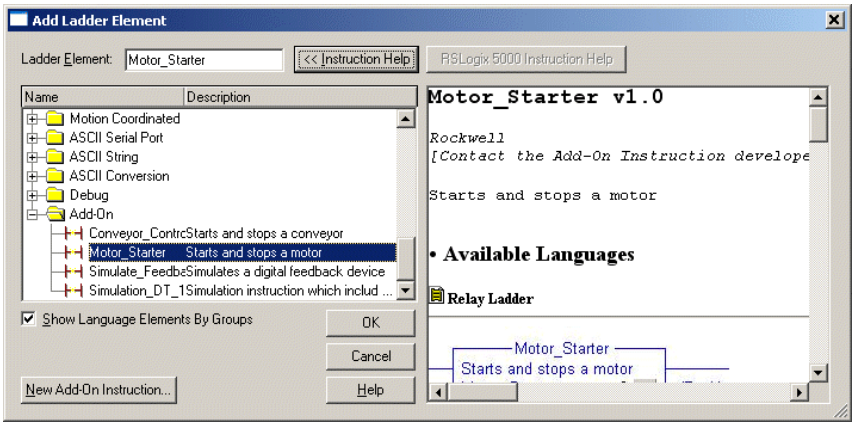


2. Locate the Add-On Instruction you want to add to your routine.



3. Select the Add-On Instruction and click OK.

   Use Instruction Help >> to display the instruction help for any instruction in the browser.



Follow this procedure when you want to use an Add-On Instruction in one of your routines.

1. Open the Add-On Instruction folder in the Controller Organizer and view the listed instructions.

   If the instruction you want to use is not listed, you need to do one of the following.

- Create the instruction in your project.

To create a new instruction, refer to Creating an Add-On Instruction on page 267.

- Copy and paste an instruction into your project.

  To copy and paste a new instruction, refer to Copying an Add-On Instruction Definition on page 327.

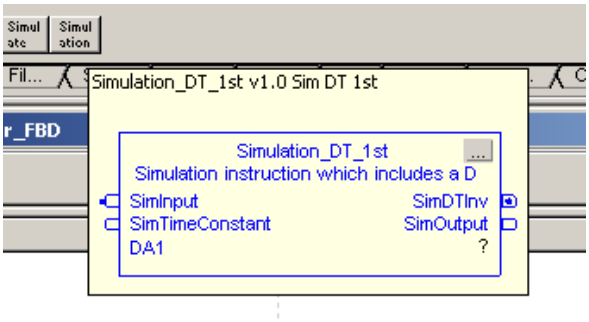
- Get the file for an exported instruction definition and then import the instruction into your current project.

  To import an instruction, refer to Importing an Add-On Instruction Definition on page 328.

2. Open the routine that will use the instruction.

3. Click the Add-On tab on the instruction toolbar.

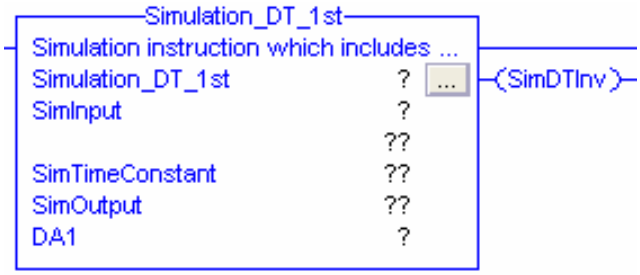
4. Click the desired Add-On Instruction, for example Simulation, and drag the instruction from the toolbar into the routine.



5. Define arguments for each Parameter on the instruction call.

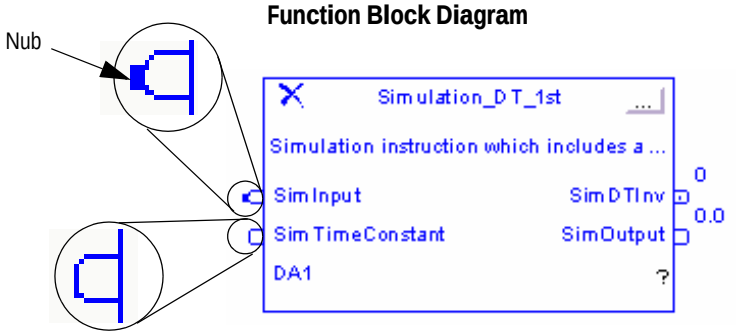
The instruction appears as follows in each of the languages.

Ladder Diagram



Parameter With	Description
Single question mark	This is a required Parameter. Enter a tag.
Single and double question marks	This is a required Parameter. Enter a tag.
Double question marks	This is not a required Parameter. You can either: • leave as is and use the default value. • if it's an input value, enter a different value.

Function Block Diagram



Item	Description
Nub on the end of pin.	This is a required Parameter. Enter a tag. You must wire the pin to an IREF, OREF, connector, or another block to verify.
Single Question Mark.	This is a required Parameter. Enter a tag.
No Nub on the end of pin.	This is not a required Parameter. You can either: • leave as is and use the default value. • if it's an input value, enter a different value.

Structured Text



The instruction expects arguments for required Parameters as listed in the instruction tooltip.

TIP For help with an instruction, select the instruction and then press [F1]. In Structured Text, make sure the cursor is in the blue instruction name.

Programming Tips when Using an Add-On Instruction

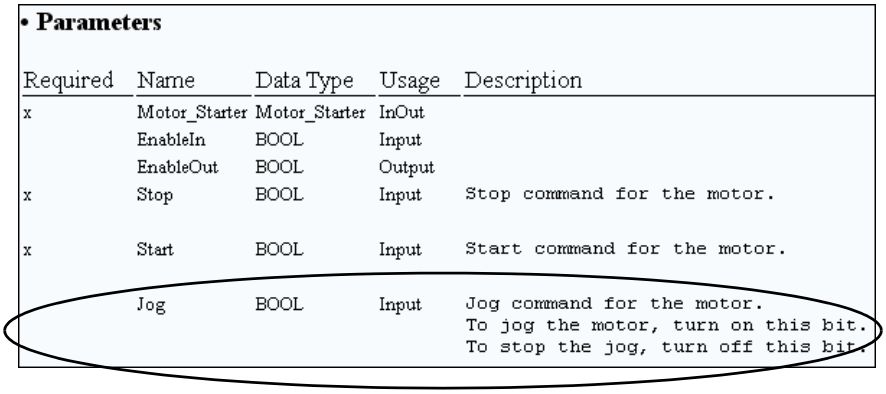
This table describes programming tips for you to reference when using Add-On Instructions.

Topic	Description
Instruction Help	Use the instruction help to determine how to use the instruction in your code.
Ladder Rungs	In a Ladder rung, consider if the instruction should be executed on a false rung condition. It may improve scan time to not execute it.
Data Types	A data type defined with the Add-On Instruction is used for the tag that provides context for the execution from your code. A tag must be defined of this type Parameter on the call to the instruction.
Indexed Tag	You can use an indirect array indexed tag for the Instruction instance. One drawback is that you cannot monitor Add-On Instruction using this as a data context.
Passing Data	• Input and Output Parameters are passed by value. • InOut Parameters are passed by reference.

Programmatically Accessing a Parameter

Follow these procedures for any language when you want to access a an Add-On Instruction Parameter that isn't available on the instruction face by default.

The following procedures demonstrate how to use the Jog parameter of the Motor Starter Add-On Instructions.

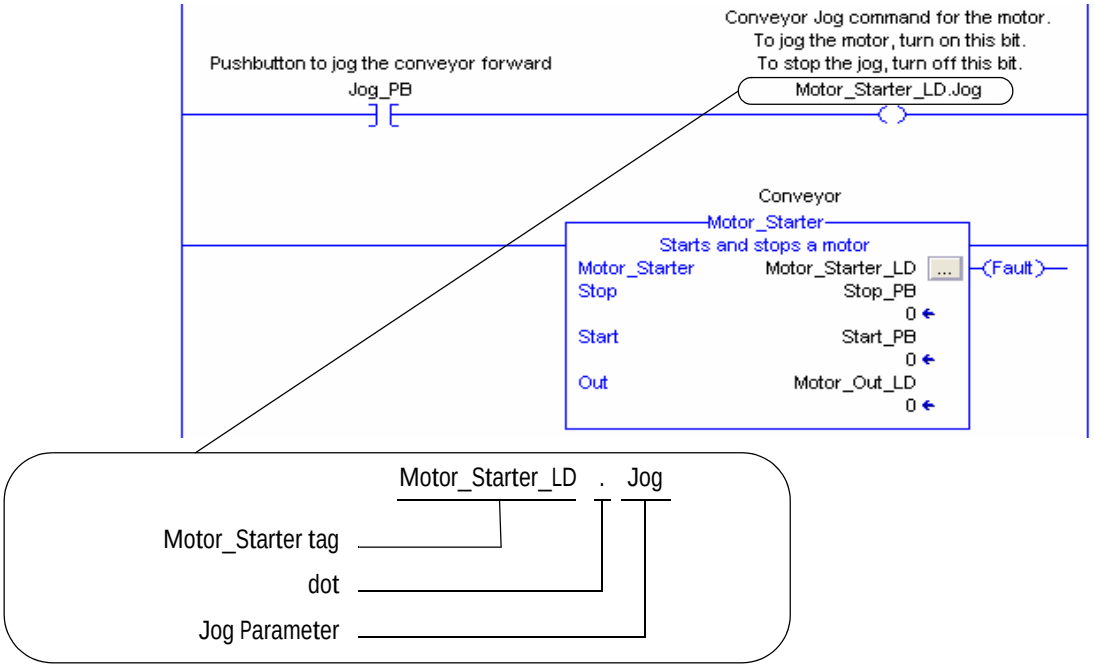


• **Parameters**

Required	Name	Data Type	Usage	Description
x	Motor_Starter	Motor_Starter	InOut	
	EnableIn	BOOL	Input	
	EnableOut	BOOL	Output	
x	Stop	BOOL	Input	Stop command for the motor.
x	Start	BOOL	Input	Start command for the motor.
	Jog	BOOL	Input	Jog command for the motor. To jog the motor, turn on this bit. To stop the jog, turn off this bit.

Using the Jog Command in Ladder Diagram

The first rung sets the Jog bit of Motor_Starter_LD = Jog_PB.



Use another instruction, an assignment, or an expression to read or write to the tag name of the Parameter. Use this format for the tag name of the Parameter.

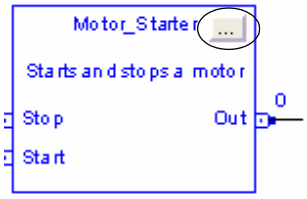
Add_On_Tag.Parameter

Where	Is
Add_On_Tag	An instance tag defined by the Add On data type.
Parameter	Name of the Parameter.

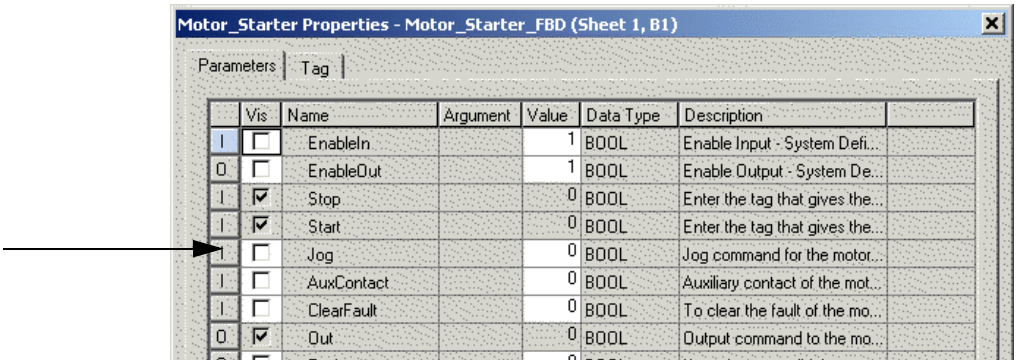
Using the Jog Command In Function Block Diagram

Any Parameter can be made visible or not except those defined as required. If the Parameter is required, you will see it checked in the Properties dialog.

1. Click Properties for the instruction.

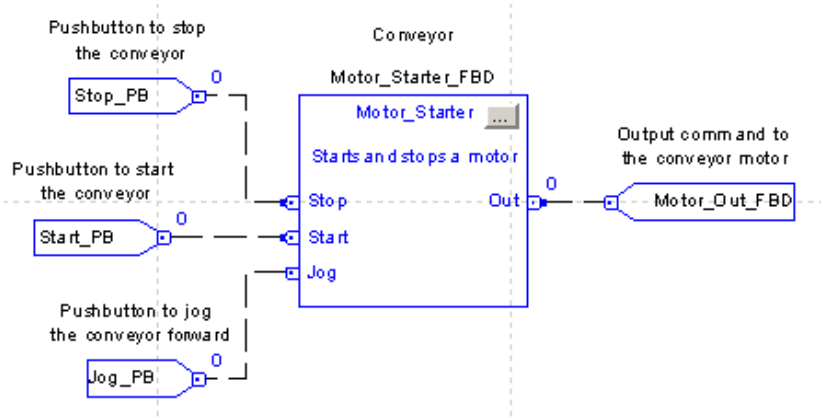


2. Check the Vis box of the Jog Parameter to use it in your diagram.

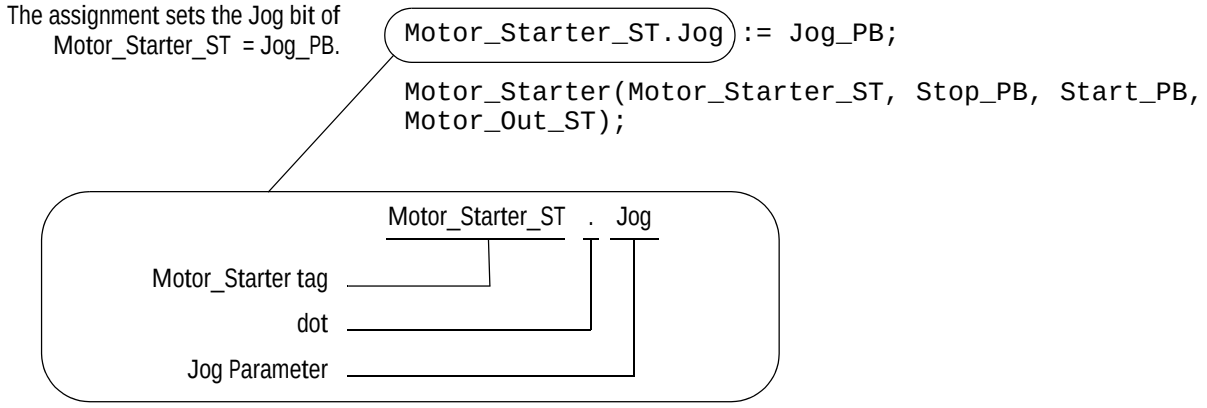


3. Click OK.

4. Wire to the pin for the Parameter.



Using the Jog Command in Structured Text

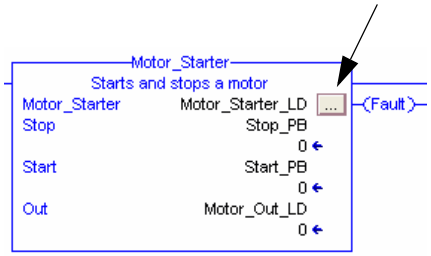


Monitoring the Value of a Parameter

Follow this procedure when you want to see or change a Parameter value of an Add-On Instruction.

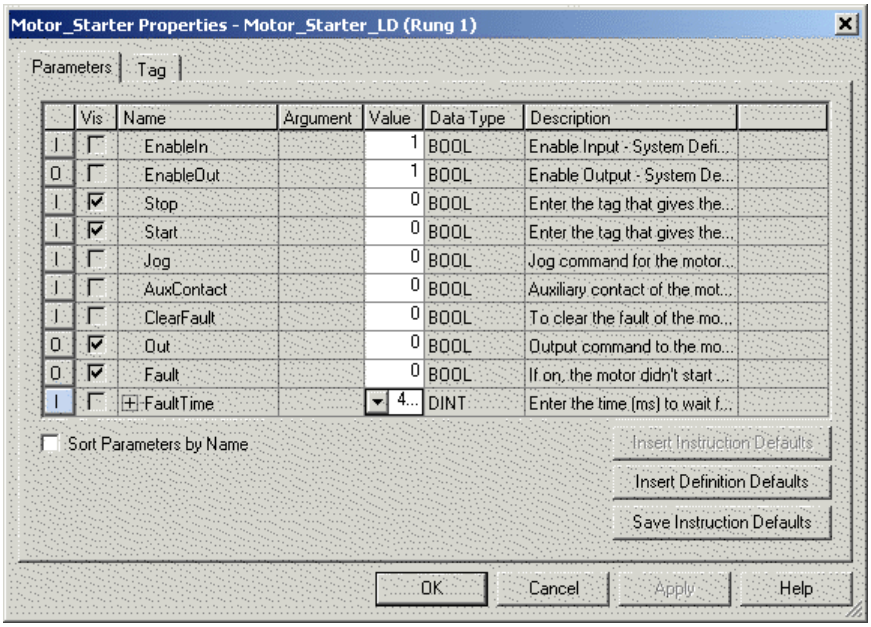
1. Open the Properties of the Instruction based on what language you are using.

- For either a Function Block or Ladder Diagram, click Properties ... for the instruction.



- For Structured Text, right-click the instruction name and choose Properties.

```
Motor_Starter(Motor_St
arter_ST,Stop_PB,Start
_PB,Motor_Out_ST);
```



2. Monitor the value of the Parameters and change any if needed.

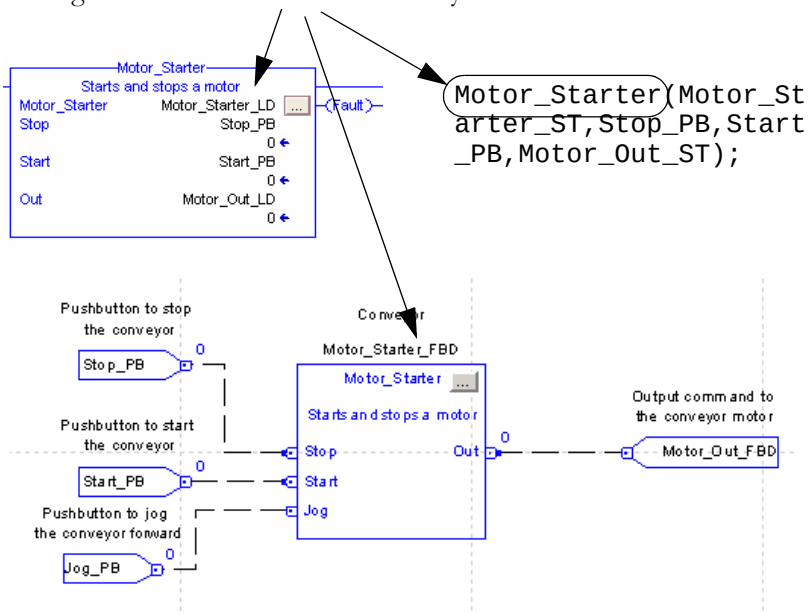
3. Type a new value for each Parameter as needed.

4. Click Apply and when finished, OK.

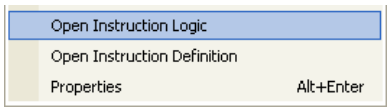
View Logic and Monitor with Data Context

Follow this procedure when you want to view the logic of an Add-On Instruction and monitor data values with the logic.

1. Right-click the instruction call in any routine.



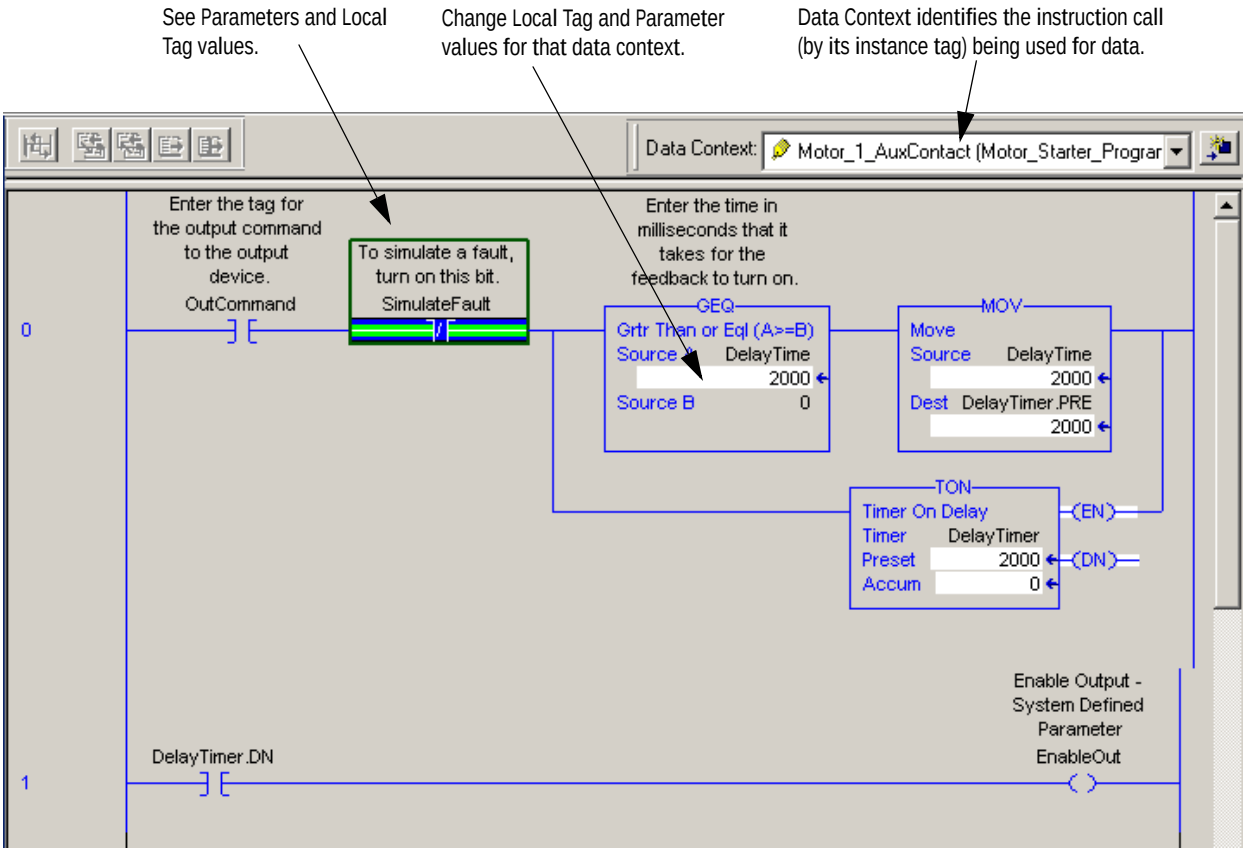
2. Select Open Instruction Logic.



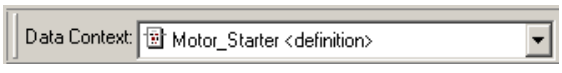
The Language Editor opens with the Add-On Instruction's Logic routine and with data values from the instruction call.

As you view the logic you can:

- identify the instruction call whose tags are being used for data.
- see the logic as it executes (when offline).
- see Parameter and Local Tag values.
- change local tag and Parameter values for the data instance selected.



To edit the logic of the Add-On Instruction, you must select the instruction <definition> in Data Context.



You can't edit logic online or when logic is in the context of an instruction call, or if the instruction is source protected.

Is the Add-On Instruction Source Protected?

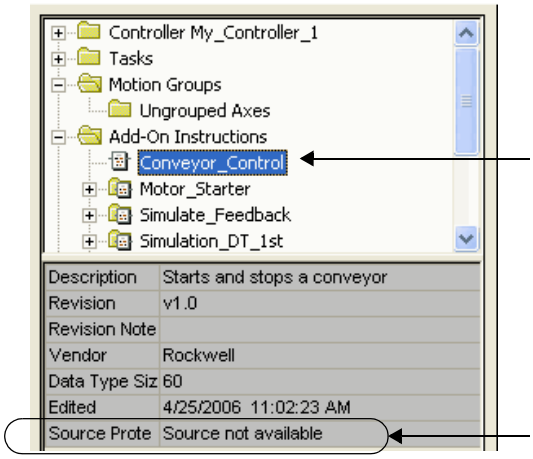
An Add-On Instruction may be source protected so you cannot view the logic. Follow these steps to see if an Add-On Instruction is source protected.

1. Select the Add-On Instruction in the Controller Organizer.

   The Add-On Instruction cannot be expanded when fully protected.

2. Look in the Quick View pane for Source Protection.

   If the Source Protection attribute isn't listed, then the instruction isn't protected.



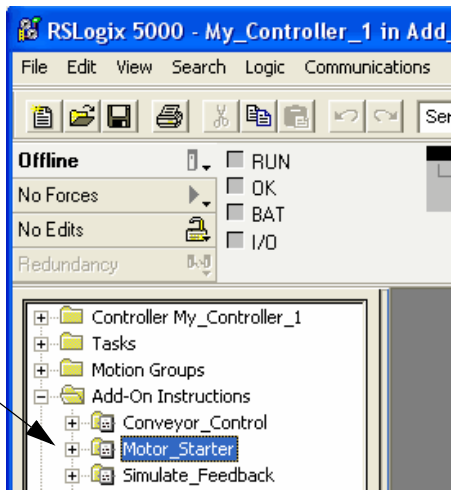
For more information about instruction protection, refer to Defining Source Protection for an Add-On Instruction on page 298.

Copying an Add-On Instruction Definition

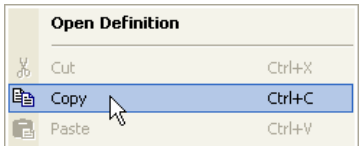
Copy an Add-On Instruction into your project when it exists in another RSLogix 5000 project. After you copy the definition, you can use the instruction as is or rename it, modify it and then use it in your programs.

1. Open the RSLogix 5000 project that contains the Add-On Instruction definition.

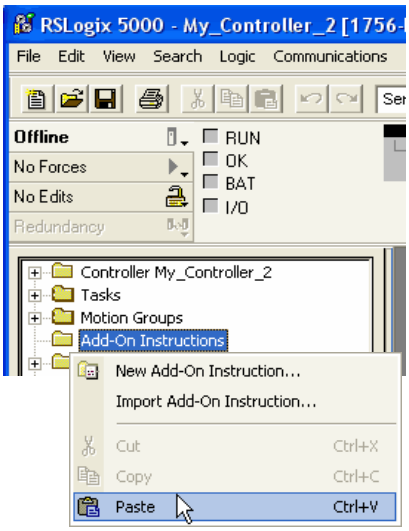
2. Find the definition in the Add-On Instructions folder.



3. Right-click the definition and choose Copy.



4. Go to your other project where you want to use the copied definition.



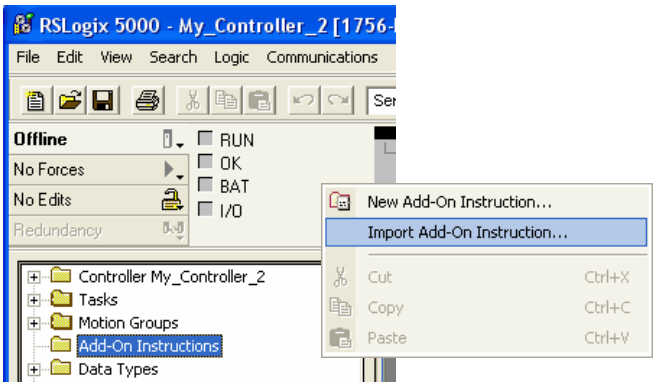
5. Right-click the Add-On Instructions folder and choose Paste.

IMPORTANT The software will add a numeric index after the Add- instruction in case the instruction's name already exi For example: Motor_Starter1.

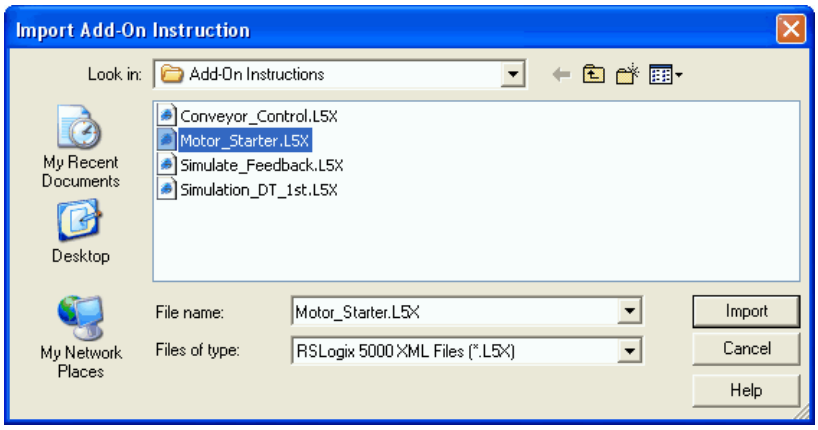
Importing an Add-On Instruction Definition

You can import a definition for an Add-On Instruction that was exported from another RSLogix 5000 project. If your project already has a revision of this Add-On Instruction, you need to refer to Updating an Add-On Instruction to a Newer Revision on page 294. Once the project has the definition, you can use the instruction in your programs.

1. Right-click the Add-On Instructions folder and select Import Add-On Instruction.



2. Locate the file containing the instruction.



3. Select the file and click Import.

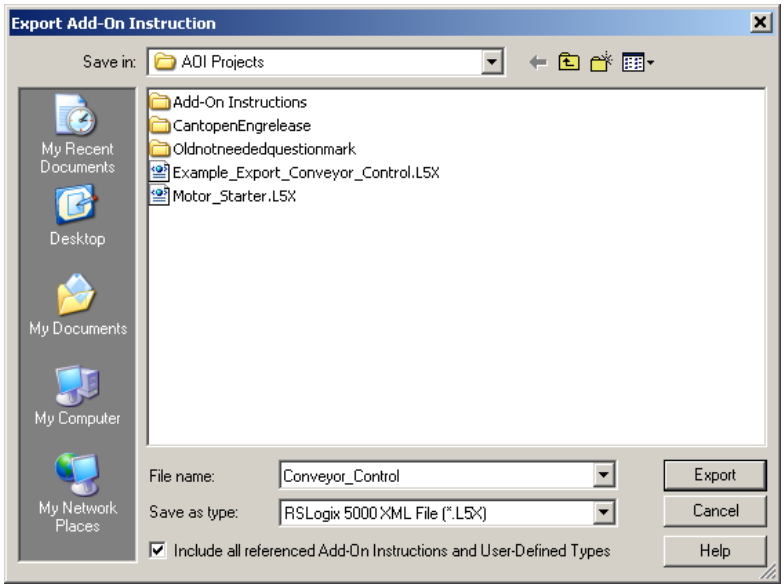
4. The instruction appears in the Controller Organizer Add-On Instruction folder.

   If the Import Configuration dialog appears, refer to Updating an Add-On Instruction to a Newer Revision on page 294.

Exporting an Add-On Instruction Definition

Follow these steps to export an Add-On Instruction.

1. Right-click Add-On Instruction in Controller Organizer and select Export Add-On Instruction.



2. Choose a location for the L5X file.
3. Type a name for the file.
4. Decide to include references or not.
5. Click Export.

Notes:

Communicating with Other Devices

When to Use This Chapter

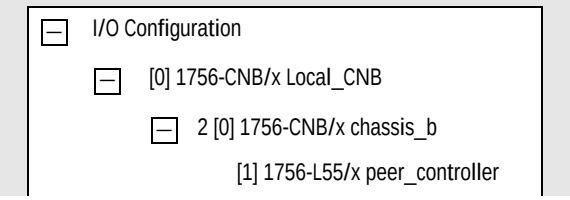
Use this chapter to plan your communication between the controller and I/O modules or other controllers.

Topic	Page
Connections	331
Produce and Consume a Tag	339
Execute a Message (MSG) Instruction	347
Get or Set the Number of Unconnected Buffers	353
Convert Between INTs and DINTs	356

Connections

A Logix5000 controller uses **connections** for many, but not all, of its communication with other devices.

Term	Definition
connection	A communication link between two devices, such as between a controller and an I/O module, PanelView terminal, or another controller. Connections are allocations of resources that provide more reliable communications between devices than unconnected messages. The number of connections that a single controller can have is limited. You indirectly determine the number of connections the controller uses by configuring the controller to communicate with other devices in the system. These communication types use connections: • I/O modules • produced and consumed tags • certain types of Message (MSG) instructions (not all types use a connection)

Term	Definition
requested packet interval (RPI)	The RPI specifies the period at which data updates over a connection. For example, an input module sends data to a controller at the RPI that you assign to the module. • Typically, you configure an RPI in milliseconds (ms). The range is 0.2 ms (200 microseconds) to 750 ms. • If a ControlNet network connects the devices, the RPI reserves a slot in the stream of data flowing across the ControlNet network. The timing of this slot may not coincide with the exact value of the RPI, but the control system guarantees that the data transfers at least as often as the RPI.
path	The path describes the route that a connection takes to get to the destination. Typically, you automatically define the path for a connection when you add the devices to the I/O Configuration folder of the controller. 

Inhibit a Connection

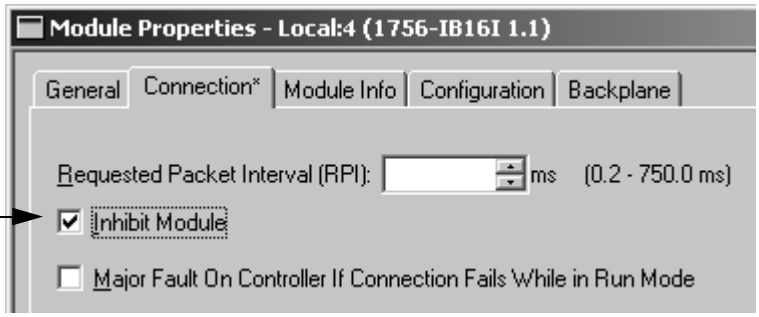
ATTENTION



Inhibiting a module breaks the connection to the module and prevents communication of I/O data.

In some situations, such as when initially commissioning a system, it is useful to disable portions of a control system and enable them as you wire up the control system. The controller lets you inhibit individual modules or groups of modules, which prevents the controller from trying to communicate with the modules.

Inhibit communication with the module. →



When you configure an I/O module, it defaults to being not inhibited. You can change an individual module's properties to inhibit a module.

If you want to	Then
communicate with the module	do not inhibit the module
prevent communication with the module	inhibit the module

When you inhibit a communication bridge module, such as a 1756-CNB or 1756-DHRIO module, the controller shuts down the connections to the bridge module and to all the modules that depend on that bridge module. Inhibiting a communication bridge module lets you disable an entire branch of the I/O network.

When you select to inhibit the module, the controller organizer displays a yellow attention symbol ⚠ over the module.

If you are	And you	And	Then
offline	→	→	The inhibit status is stored in the project. When you download the project, the module is still inhibited.
online	inhibit a module while you are connected to the module	→	The connection to the module is closed. The modules' outputs go to the last configured program mode.
	inhibit a module but a connection to the module was *not* established (perhaps due to an error condition or fault)	→	The module is inhibited. The module status information changes to indicate that the module is inhibited and not faulted.
	uninhibit a module (clear the check box)	no fault occurs	A connection is made to the module and the module is dynamically reconfigured (if the controller is the owner controller) with the configuration you created for that module. If the controller is configured for listen-only, it cannot reconfigure the module.
		fault occurs	A connection is *not* made to the module. The module status information changes to indicate the fault condition.

To inhibit or uninhibit a module from logic:

1. Use a Get System Value (GSV) instruction to read the Mode attribute for the module.
2. Set or clear bit 2:

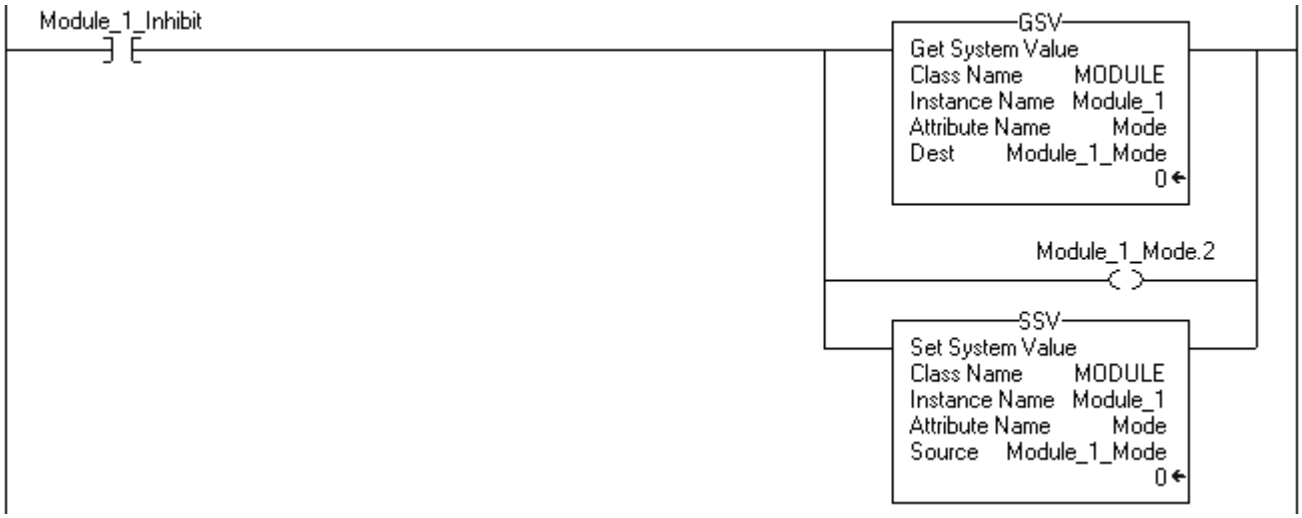
If you want to	Then
inhibit the module	Set bit 2.
uninhibit the module	Clear bit 2

3. Use a Set System Value (SSV) instruction to write the Mode attribute back to the module.

EXAMPLE Inhibit a Connection

If Module_1_Inhibit = 1, then inhibit the operation of the I/O module named Module_1:

1. The GSV instruction sets Module_1_Mode = value of the Mode attribute for the module.
2. The OTE instruction sets bit 2 of Module_1_Mode = 1. This means inhibit the connection.
3. The SSV instruction sets the Mode attribute for the module = Module_1_Mode.



Manage a Connection Failure

ATTENTION

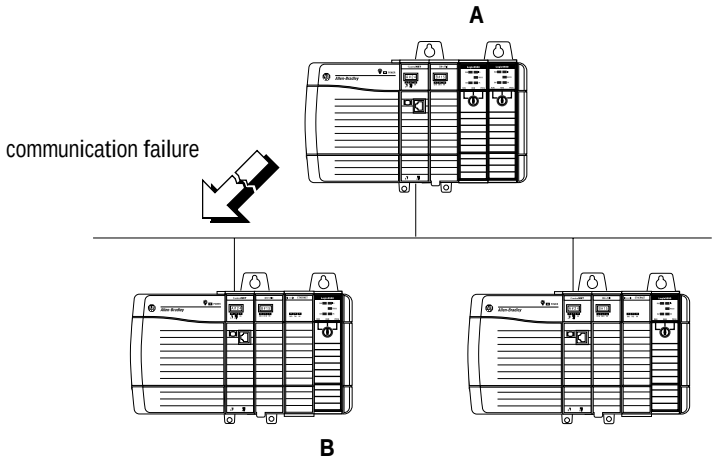


Outputs respond to the last, non-faulted state of the controlling inputs. To avoid potential injury and damage to machinery, make sure this does not create unsafe operation. Configure critical I/O modules to generate a controller major fault when they lose their connections to the controller. Or, monitor the status of I/O modules.

If the controller loses communication with a module, data from that device does not update. When this occurs, the logic makes decisions on data that may or may not be correct.

EXAMPLE Loss of communication

Controller B requires data from controller A. If communication fails between the controllers, then controller B continues to act on the last data that it received from controller A.



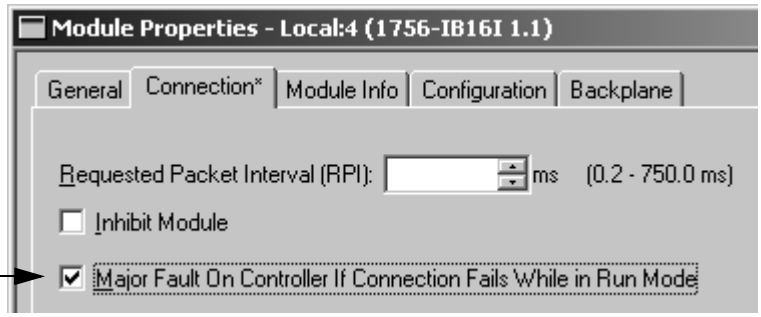
If communication with a device in the I/O configuration of the controller does not occur for 100 ms, the communication times out. If this occurs, you have these options:

If you want the controller to	Then
fault (major fault)	Configure a Major Fault to Occur
continue operating	Monitor the Health of a Module

Configure a Major Fault to Occur

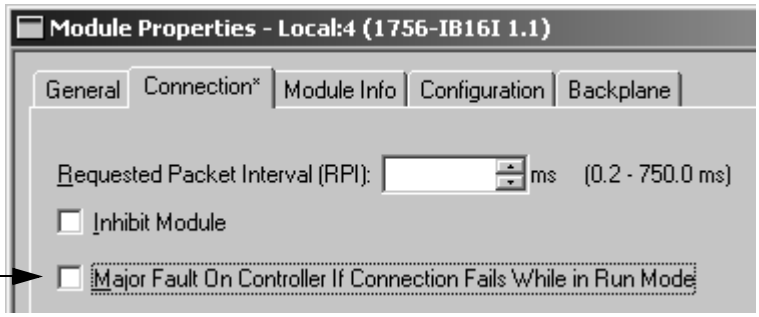
You can configure modules to generate a major fault in the controller if they lose their connection with the controller. This interrupts the execution of logic and executes the Controller Fault Handler. If the Controller Fault Handler does not clear the fault, then the controller shuts down.

If the connection times out, produce a major fault in the controller.



Monitor the Health of a Module

If you do not configure the major fault to occur, you should monitor the module status. If a module loses its connection to the controller, outputs go to their configured faulted state. The controller and other I/O modules continue to operate based on old data from the module.



If the connection times out, continue operation without invoking a major fault on the controller.

If communication with a module times out, the controller produces these warnings:

- The I/O LED on the front of the controller flashes green.
- A ⚠ shows over the I/O configuration folder and over the device (s) that has timed out.
- A module fault code is produced, which you can access through:
  - Module Properties window for the module
  - GSV instruction

To monitor the health of your connections, use a Get System Value (GSV) instruction to monitor the MODULE object for either the controller or a specific module:

If you want to	Get this attribute	Data Type	Description	
determine if communication has timed out with any device	LEDStatus	INT For efficiency, use a DINT as the destination data type.	Specifies the current state of the I/O LED on the front of the controller. **Note:** You do not enter an instance name with this attribute. This attribute applies to the entire collection of modules.	
			Value	**Meaning**
			0	**LED off:** No MODULE objects are configured for the controller (there are no modules in the I/O Configuration section of the controller organizer).
			1	Flashing red: None of the MODULE objects are Running.
			2	Flashing green: At least one MODULE object is not Running.
			3	Solid green: All the Module objects are Running.
determine if communication has timed out with a specific device	FaultCode	INT For efficiency, use a DINT as the destination data type.	A number which identifies a module fault, if one occurs. In the Instance Name, choose the device whose connection you want to monitor. Make sure to assign a name to the device in the I/O Configuration folder of the project.	

EXAMPLE Monitor the Health of a Module

The GSV instruction continuously sets I_O_LED_Status (DINT tag) = status of the I/O LED of the controller.



If I_O_LED_Status = 2, then communication has timed out (faulted) with at least one module. The GSV instruction sets Module_3_Fault_Code = fault code for Module_3.



If Module_3_Fault_Code. is NOT equal to 0, then communication has timed out (faulted) with Module_3. The OTE instruction sets Module_3_Faulted = 1.

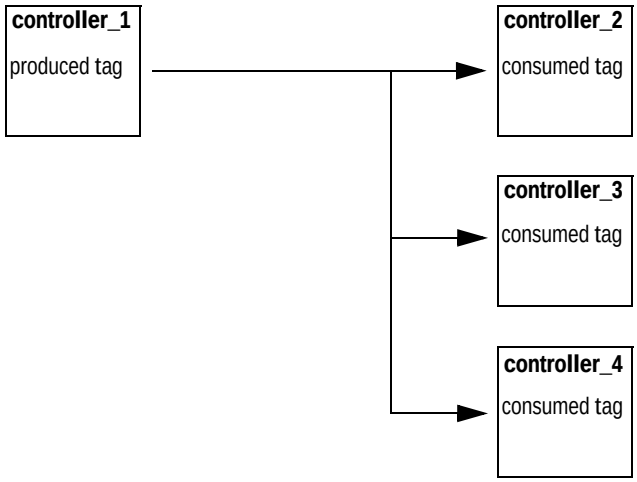


Produce and Consume a Tag

To transfer data between controllers (send or receive data), you can:

If the data	Then	See
needs regular delivery at an interval that you specify (i.e., deterministic)	Produce and Consume a Tag	This section
is sent when a specific condition occurs in your application	Execute a Message (MSG) Instruction	Page 11-347

A Logix5000 controller lets you produce (broadcast) and consume (receive) system-shared tags.



Term	Definition
produced tag	A tag that a controller makes available for use by other controllers. Multiple controllers can simultaneously consume (receive) the data. A produced tag sends its data to one or more consumed tags (consumers) without using logic. The produced tag sends its data at the RPI of the consuming tag.
consumed tag	A tag that receives the data of a produced tag. The data type of the consumed tag must match the data type (including any array dimensions) of the produced tag. The RPI of the consumed tag determines the period at which the data updates.

Controllers and Networks that Support Produced/Consumed Tags

These controller and network combinations support produced and consumed tags.

This controller	Can produce and consume tags over this network		
	Backplane	ControlNet	EtherNet/IP
SLC 500		4	
PLC-5		4	
CompactLogix(1)		4	4
ControlLogix	4	4	4
DriveLogix		4	4
FlexLogix		4	4
SoftLogix5800		4	

(1) For ControlNet, use a 1769-L32C or 1769-L35CR controller. For EtherNet/IP, use a 1769-L32Eor 1769-L35E controller.

For two controllers to share produced or consumed tags, both controllers must be attached to the same network (such as a ControlNet or Ethernet/IP network). You cannot bridge produced and consumed tags over two networks.

Connection Requirements of a Produced or Consumed Tag

IMPORTANT If a consumed-tag connection fails, all of the other tags being consumed from that remote controller stop receiving new data.

Produced and consumed tags each require connections. As you increase the number of controllers that can consume a produced tag, you also reduce the number of connections the controller has available for other operations, like communications and I/O.

Each produced or consumed tag uses these connections:

And this type of tag	Uses this many connections
produced tag	*number_of_consumers* + 1
consumed tag	1

EXAMPLE Connection Requirements of a Produced or Consumed Tag

- A FlexLogix controller producing a tag for 5 controllers (consumers) uses 6 connections.
- A ControlLogix controller producing 4 tags for 1 controller uses 8 connections:
  - Each tag uses 2 connections (1 consumer + 1 = 2).
  - 2 connections per tag x 4 tags = 8 connections
- Consuming 4 tags from a controller uses 4 connections (1 connection per tag x 4 tags = 4 connections).

Organize Tags for Produced or Consumed Data

As you organize your tags for produced or consumed data (shared data), follow these guidelines:

Guideline	Details			
Create the tags at the **controller scope**.	You can share only controller-scoped tags.			
Use one of these data types: • DINT • REAL • array of DINTs or REALs • user-defined	• To share other data types, create a user-defined data type that contains the required data. • Use the same data type for the produced tag and corresponding consumed tag or tags.			
To share tags with a PLC-5C controller, use a user-defined data type.	**To**	**This**	**Then**	
	produce	integers	Create a user-defined data type that contains an array of INTs with an even number of elements, such as INT[2]. (When you produce INTs, you must produce two or more.)	
		only one REAL value	Use the REAL data type.	
		more than one REAL value	Create a user-defined data type that contains an array of REALs.	
	consume	integers	Create a user-defined data type that contains these members:	
			Data type:	**Description:**
			DINT	Status
			INT[*x*], where *x* is the output size of the data from the PLC-5C controller. (If you are consuming only one INT, omit *x*.)	Data produced by a PLC-5C controller
Limit the size of the tag to ≤ 500 bytes.	• If you must transfer more than 500 bytes, create logic to transfer the data in packets. See chapter 12. • If you produce the tag over a ControlNet network, the tag may need to be less than 500 bytes. Refer to "Adjust for Bandwidth Limitations" on page 11-342.			

Guideline	Details
Use the highest permissible RPI for your application.	If the controller consumes the tag over a ControlNet network, use a binary multiple of the ControlNet network update time (NUT). For example, if the NUT is 5 ms, use an RPI of 5, 10, 20, 40 ms, etc.
Combine data this goes to the same controller.	If you are producing several tags for the same controller: • Group the data into one or more user-defined data types. (This uses less connections than producing each tag separately.) • Group the data according to similar update intervals. (To conserve network bandwidth, use a greater RPI for less critical data.) For example, you could create one tag for data that is critical and another tag for data that is not as critical.

Adjust for Bandwidth Limitations

When you share a tag over a ControlNet network, the tag must fit within the bandwidth of the network:

- As the number of connections over a ControlNet network increases, several connections, including produced or consumed tags, may need to share a network update time (NUT).
- Since a ControlNet network can only pass 500 bytes in one NUT, the data of each connection must be less then 500 bytes to fit into the NUT.

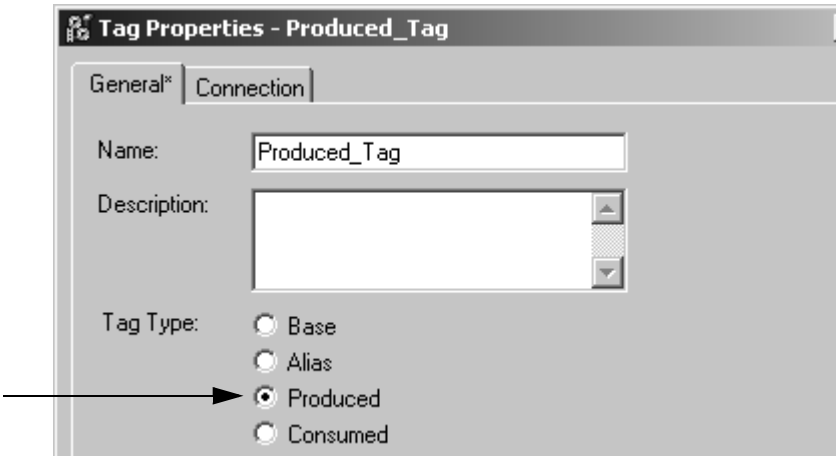
Depending on the size of your system, you may not have enough bandwidth on your ControlNet network for a tag of 500 bytes. If a tag is too large for your ControlNet network, make one or more of these adjustments:

Adjustment	Description	
Reduce your network update time (NUT).	At a faster NUT, less connections have to share an update slot.	
Increase the requested packet interval (RPI) of your connections.	At higher RPIs, connections can take turns sending data during an update slot.	
For a ControlNet bridge module (CNB) in a remote chassis, select the most efficient communication format for that chassis:	**Are most of the modules in the chassis non-diagnostic, digital I/O modules?**	**Then select this communication format for the remote CNB module:**
	Yes	Rack Optimization
	No	None
	The Rack Optimization format uses an additional 8 bytes for each slot in its chassis. Analog modules or modules that are sending or getting diagnostic, fuse, timestamp, or schedule data require direct connections and cannot take advantage of the rack optimized form. Selecting “None” frees up the 8 bytes per slot for other uses, such as produced or consumed tags.	
Separate the tag into two or more smaller tags.	1. Group the data according to similar update rates. For example, you could create one tag for data that is critical and another tag for data that is not as critical. 2. Assign a different RPI to each tag.	
Create logic to transfer the data in smaller sections (packets).	Refer to "Producing a Large Array" on page 12-359.	

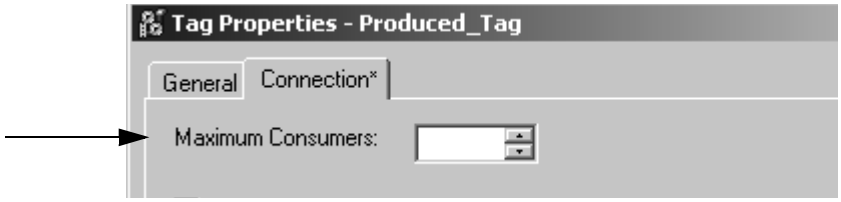
Produce a Tag



1. In the controller organizer, right-click the Controller Tags folder and choose Edit Tags. (You can produce only controller-scoped tags.)
2. In the Controller Tags window, right-click the tag that you want to produce and choose Edit Tag Properties.
3. Click the Produced option button.



4. Select the Connection tab and specify the number of controllers that will consume (receive) the tag.



5. Click OK

Consume Data That Is Produced by Another Controller

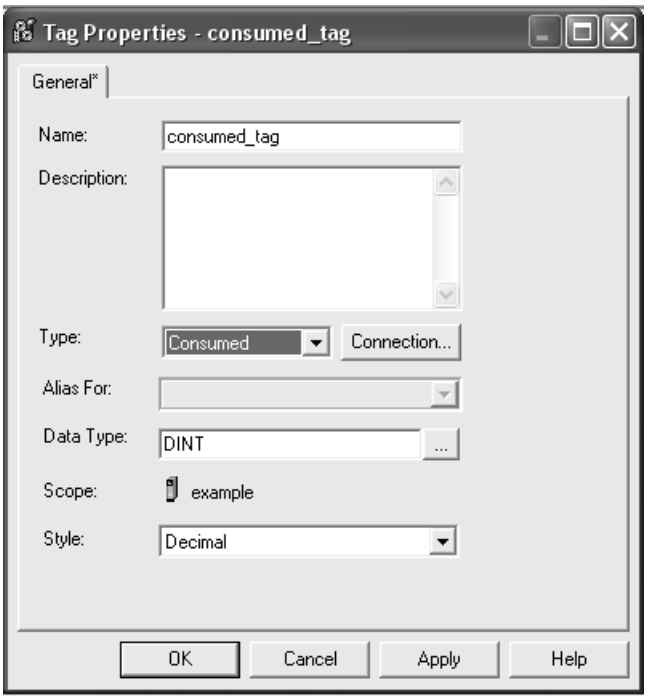
1. In the controller organizer, I/O Configuration folder, add the controller that is producing the data (the other Logix5000 controller or PLC-5C controller).



2. In the controller organizer, right-click the Controller Tags folder and choose Edit Tags. (Only controller-scoped tags can consume data.)

3. In the Controller Tags window, right-click the tag that will consume the data and choose Edit Tag Properties.

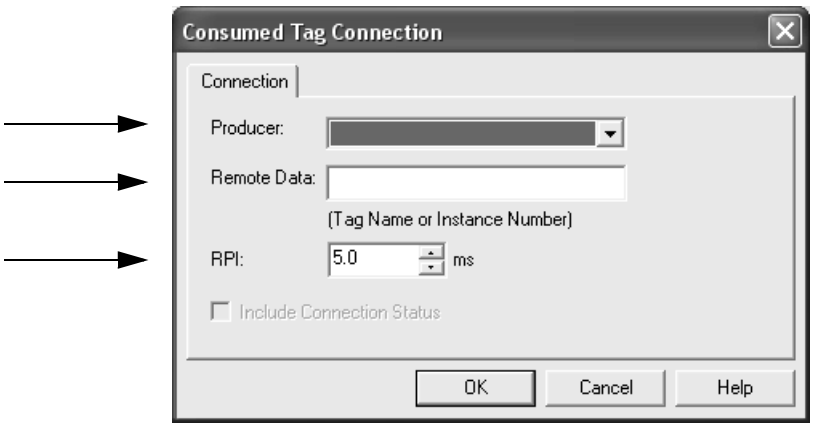
4. Select Consumed for the type.



5. Specify the data type:

If the producing controller is	Then the data type should be	
Logix5000 controller	same data type as the produced tag	
PLC-5C controller	user-defined data type with these members:	
	Data type	**Description**
	DINT	Status
	INT[*x*], where *x* is the output size of the data from the PLC-5C controller. (If you are consuming only one INT, omit *x*.)	Data produced by a PLC-5C controller

6. Click the Connection buttong to define the consumed tag.



- Select the controller that produces the data.
- Type the name or instance number of the remote, produced data.

If the producing controller is	Then type or select
Logix5000 controller	tag name of the produced tag
PLC-5C controller	message number from the ControlNet configuration of the PLC-5C controller

- Type or select the requested packet interval (RPI) for the connection.

7. Click OK.

8. If you consume the tag over a ControlNet network, use RSNetWorx for ControlNet software to schedule the network.

Additional Steps for a PLC-5C Controller

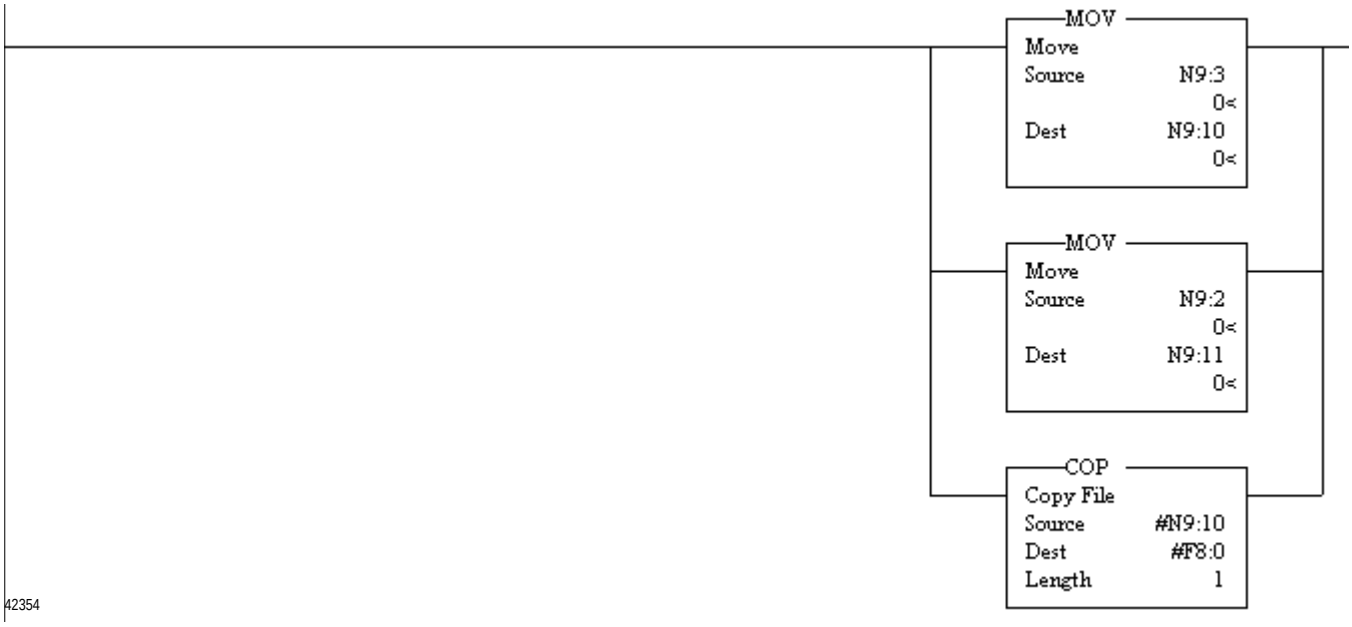
If you are sharing data with a PLC-5C controller, you need to:

Action	Details		
In the ControlNet configuration of the PLC-5C controller, scheduled a message.	**If the PLC-5C**	**This**	**Then in RSNetWorx software**
	produces	integers	In the ControlNet configuration of the PLC-5C controller, insert a Send Scheduled Message.
	consumes	integers	In the ControlNet configuration of the PLC-5C controller: A. Insert a Receive Scheduled Message. B. In the Message size, enter the number of integers in the produced tag.
		REALs	In the ControlNet configuration of the PLC-5C controller: A. Insert a Receive Scheduled Message. B. In the Message size, enter two times the number of REALs in the produced tag. For example, if the produced tag contains 10 REALs, enter 20 for the Message size.
If the PLC-5C controller consumes REALs, reconstruct the values.	When you produce REALs (32-bit floating-point values) for a PLC-5C controller, the PLC-5C stores the data in consecutive 16-bit integers: • The first integer contains the upper (left-most) bits of the value. • The second integer contains the lower (right-most) bits of the value. • This pattern continues for each floating-point value. See the example on page 11-347.		

This example shows how to re-construct a REAL (floating point value) in the PLC-5C controller

EXAMPLE Re-construct a floating point value

The two MOV instructions reverse the order of the integers as the integers move to a new location. Because the destination of the COP instruction is a floating-point address, it takes two consecutive integers, for a total of 32 bits, and converts them to a single floating-point value.



Execute a Message (MSG) Instruction

To transfer data between controllers (send or receive data), you can:

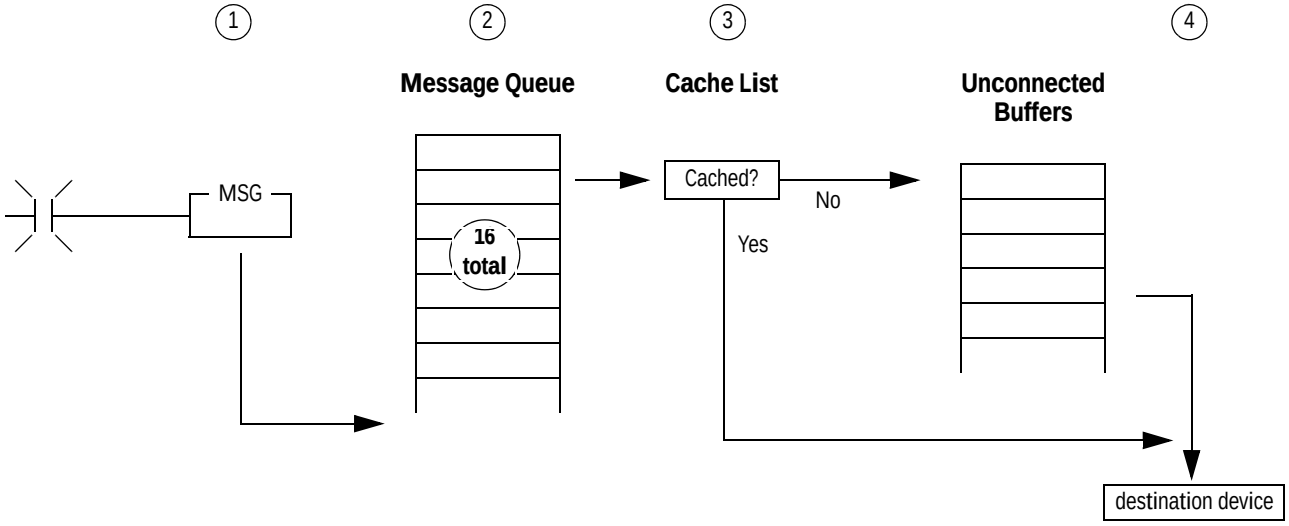
If the data	Then	See
needs regular delivery at a rate that you specify (i.e., deterministic)	Produce and Consume a Tag	Page 11-339
is sent when a specific condition occurs in your application	Execute a Message (MSG) Instruction	This section

EXAMPLE Execute a Message (MSG) Instruction

If count_send = 1 and count_msg.EN = 0 (MSG instruction is not already enabled), then execute a MSG instruction that sends data to another controller.



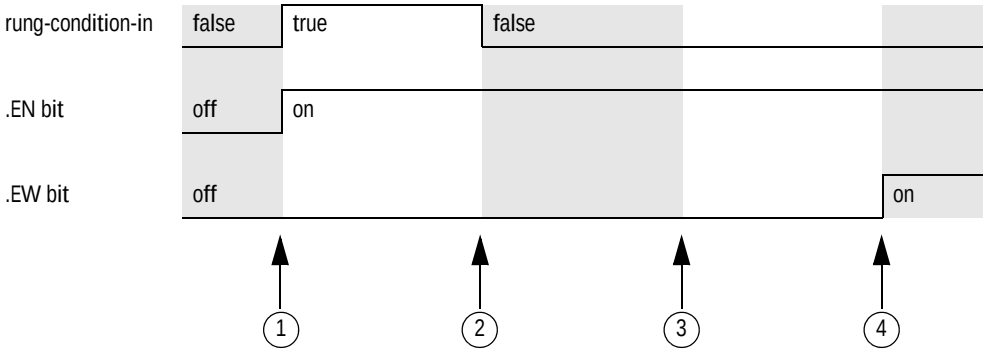
This diagram shows how the controller processes Message (MSG) instructions.



Description		
1	The controller scans the MSG instruction and its rung-condition-in goes true. The MSG instruction enters the message queue.	
2	The MSG instruction comes off the queue and is processed.	
3	**If the MSG instruction**	**Then the MSG instruction**
	does not use a connection or the connection was **not** previously cached.	uses an unconnected buffer to establish communication with the destination device
	uses a connection and the connection is cached	does not use an unconnected buffer
4	Communication occurs with the destination device.	

Message Queue

The message queue holds up to 16 MSG instructions, including those that you configure as a block-transfer read or block-transfer write. When the queue is full, an instruction tries to enter the queue on each subsequent scan of the instruction, as shown below:



	Description
1	The controller scans the MSG instruction. The rung-condition-in for the MSG instruction is true. The EN bit is set. The MSG instruction attempts to enter the queue but the queue is full (16 MSG instructions are already enabled). The EW bit remains cleared.
2 & 3	The controller scans the MSG instruction. The rung-condition-in for the MSG instruction is false. The EN bit remains set. The MSG instruction attempts to enter the queue but the queue is full. The EW bit remains cleared.
4	The controller scans the MSG instruction. The MSG instruction attempts to enter the queue. The queue has room so the instruction enters the queue. The EW bit is set.

Cache List

Depending on how you configure a MSG instruction, it may use a connection to send or receive data.

This type of message	And this communication method:	Uses a connection
CIP data table read or write	→	3
PLC2, PLC3, PLC5, or SLC (all types)	CIP	
	CIP with Source ID	
	DH+	3
CIP generic	→	your option (1)
block-transfer read or write	→	3

(1) You can connect CIP generic messages. But for most applications we recommend you leave CIP generic messages unconnected.

If a MSG instruction uses a connection, you have the option to leave the connection open (cache) or close the connection when the message is done transmitting.

If you	Then
Cache the connection	The connection stays open after the MSG instruction is done. This optimizes execution time. Opening a connection each time the message executes increases execution time.
Do not cache the connection	The connection closes after the MSG instruction is done. This frees up that connection for other uses.

The controller has these limits on the number of connections that you can cache:

If you have this software and firmware revision	Then you can cache
11.x or earlier	• block transfer messages for up to 16 connections • other types of messages for up to 16 connections
12.x or later	up to 32 connections

If several messages go to the same device, the messages may be able to share a connection.

If the MSG instructions are to	And they are	Then
different devices	→	Each MSG instruction uses 1 connection.
same device	enabled at the same time	Each MSG instruction uses 1 connection.
	NOT enabled at the same time	The MSG instructions share the connection. (For example, together they count as 1 connection.)

EXAMPLE Share a Connection

If the controller alternates between sending a block-transfer read message and a block-transfer write message to the same module, then together both messages count as 1 connection. Caching both messages counts as 1 on the cache list.

Unconnected Buffers

To establish a connection or process unconnected messages, the controller uses an unconnected buffer.

Term	Definition
unconnected buffer	An allocation of memory that the controller uses to process unconnected communication. The controller performs unconnected communication when it: • establishes a connection with a device, including an I/O module • executes a MSG instruction that does not use a connection The controller can have 10 - 40 unconnected buffers. • The default number is 10. • To increase the number of unconnected buffers, execute a MSG instruction that reconfigures the number of unconnected buffers. Refer to Get or Set the Number of Unconnected Buffers on page 11-353. • Each unconnected buffers uses 1.1K bytes of memory. • If all the unconnected buffers are in use when an instruction leaves the message queue, the instruction errors and data does not transfer.

If a MSG instruction uses a connection, the instruction uses an unconnected buffer when it first executes to establish a connection. If you configure the instruction to cache the connection, it no longer requires an unconnected buffer once the connection is established.

Guidelines

As you plan and program your MSG instructions, follow these guidelines:

Guideline	Details
1. For each MSG instruction, create a control tag.	Each MSG instruction requires its own control tag. • Data type = MESSAGE • Scope = controller • The tag cannot be part of an array or a user-defined data type.
2. Keep the source and/or destination data at the controller scope.	A MSG instruction can access only tags that are in the Controller Tags folder (controller scope).
3. If your MSG is to a device that uses 16-bit integers, use a buffer of INTs in the MSG and DINTs throughout the project.	If your message is to a device that uses 16-bit integers, such as a PLC-5 or SLC 500 controller, and it transfers integers (not REALs), use a buffer of INTs in the message and DINTs throughout the project. This increases the efficiency of your project because Logix5000 controllers execute more efficiently and use less memory when working with 32-bit integers (DINTs). Refer to Convert Between INTs and DINTs on page 11-356.
4. Cache the connected MSGs that execute most frequently.	Cache the connection for those MSG instructions that execute most frequently, up to the maximum number permissible for your controller revision. This optimizes execution time because the controller does not have to open a connection each time the message executes.
5. If you want to enable more than 16 MSGs at one time, use some type of management strategy.	If you enable more than 16 MSGs at one time, some MSG instructions may experience delays in entering the queue. To guarantee the execution of each message, use one of these options: • Enable each message in sequence. • Enable the messages in groups. • Program a message to communicate with multiple devices. For more information, see Appendix B. • Program logic to coordinate the execution of messages. For more information, see Appendix A.
6. Keep the number of unconnected and uncached MSGs less than the number of unconnected buffers.	The controller can have 10 - 40 unconnected buffers. The default number is 10. • If all the unconnected buffers are in use when an instruction leaves the message queue, the instruction errors and does not transfer the data. • You can increase the number of unconnected buffers (40 max.), but continue to follow guideline 5. • To increase the number of unconnected buffers, see page 11-353.

Get or Set the Number of Unconnected Buffers

To determine or change the number of unconnected buffers, use a MSG instruction.

- The range is 10 - 40 unconnected buffers.
- The default number is 10.
- Each unconnected buffers uses 1.1K bytes of memory.

Get the Number of Unconnected Buffers

To determine the number of unconnected buffers that the controller currently has available, configure a Message (MSG) instruction as follows:

On this tab	For this item	Type or select
Configuration	Message Type	CIP Generic
	Service Type	Custom
	Service Code	3
	Class	304
	Instance	1
	Attribute	0
	Source Element	*source_array* where data type = SINT[4]
		In this element / **Enter**: *source_array*[0] = 1; *source_array*[1] = 0; *source_array*[2] = 17; *source_array*[3] = 0
	Source Length (bytes)	4 (Write 4 SINTs.)
	Destination	*destination_array* where data type = SINT[10] (Leave all the values = 0.)
		destination_array[6] = current number of unconnected buffers
Communication	Path	1, *slot_number_of_controller*

Set the Number of Unconnected Buffers

As a starting value, set the number of unconnected buffers equal to the number of unconnected and uncached messages enabled at one time plus approximately 5. The additional 5 buffers provides a cushion in case you underestimate the number of messages that are enabled at one time.

To change the number of unconnected buffers of the controller, configure a Message (MSG) instruction as follows:

On this tab	For this item	Type or select
Configuration	Message Type	CIP Generic
	Service Type	Custom
	Service Code	4
	Class	304
	Instance	1
	Attribute	0
	Source Element	*source_array* where data type = SINT[8]
	Source Length (bytes)	8 (Write 8 SINTs.)
	Destination	*destination_array* where data type = SINT[6] (Leave all the values = 0.)
Communication	Path	1, *slot_number_of_controller*

In this element	Enter
source_array[0]	1
source_array[1]	0
source_array[2]	17
source_array[3]	0
source_array[4]	Number of unconnected buffers that you want.
source_array[5]	0
source_array[6]	0
source_array[7]	0

EXAMPLE Set the Number of Unconnected Buffers

If S:FS = 1 (first scan), then set the number of unconnected buffers for the controller:

Source_Array[0] = 1

Source_Array[1] = 0

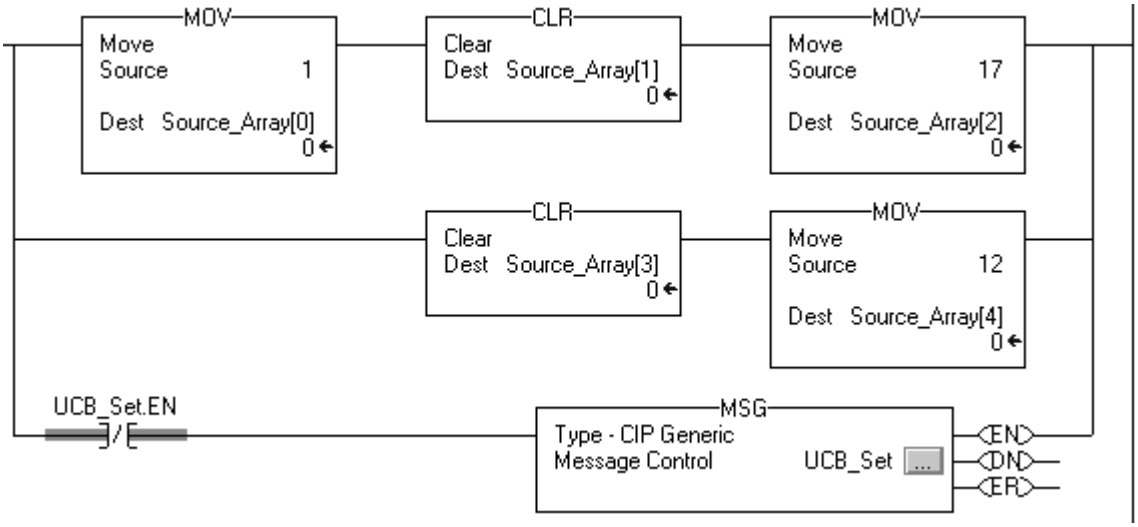
Source_Array[2] = 17

Source_Array[3] = 0

Source_Array[4] = 12 (The number of unconnected buffers that you want. In this example, we want 12 buffers.)

If UCB_Set.EN = 0 (MSG instruction is not already enabled) then:

MSG instruction sets the number of unconnected buffers = Source_Array[4]



where:

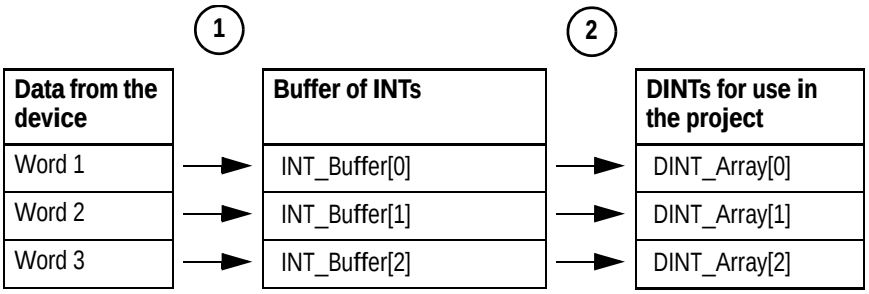
Tag Name	Type	Description
UCB_Set	MESSAGE	Control tag for the MSG instruction.
Source_Array	SINT[8]	Source values for the MSG instruction, including the number of unconnected buffers that you want.

Convert Between INTs and DINTs

In the Logix5000 controller, use the **DINT** data type for integers whenever possible. Logix5000 controllers execute more efficiently and use less memory when working with 32-bit integers (DINTs).

If your message is to a device that uses 16-bit integers, such as a PLC-5 or SLC 500 controller, and it transfers integers (not REALs), use a buffer of INTs in the message and DINTs throughout the project. This increases the efficiency of your project.

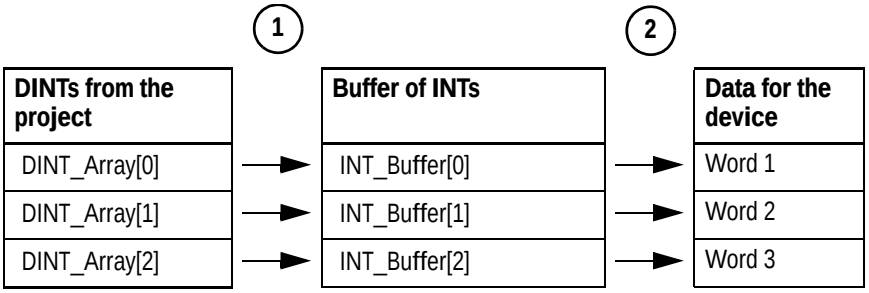
Read 16-Bit Integers



1. The Message (MSG) instruction reads 16-bit integers (INTs) from the device and stores them in a temporary array of INTs.

2. An File Arith/Logical (FAL) instruction converts the INTs to DINTs for use by other instructions in your project.

Write 16-Bit Integers



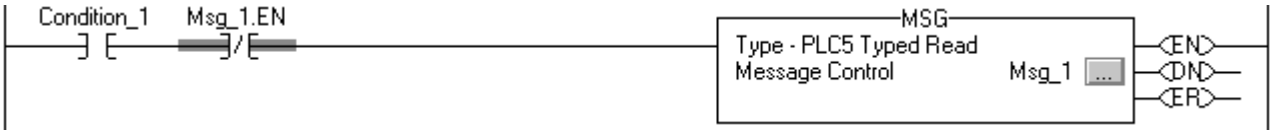
1. An FAL instruction converts the DINTs from the Logix5000 controller to INTs.

2. The MSG instruction writes the INTs from the temporary array to the device.

EXAMPLE Read integer values from a PLC-5 controller

If Condition_1 = 1 And Msg_1.EN = 0 (MSG instruction is not already enabled) then:

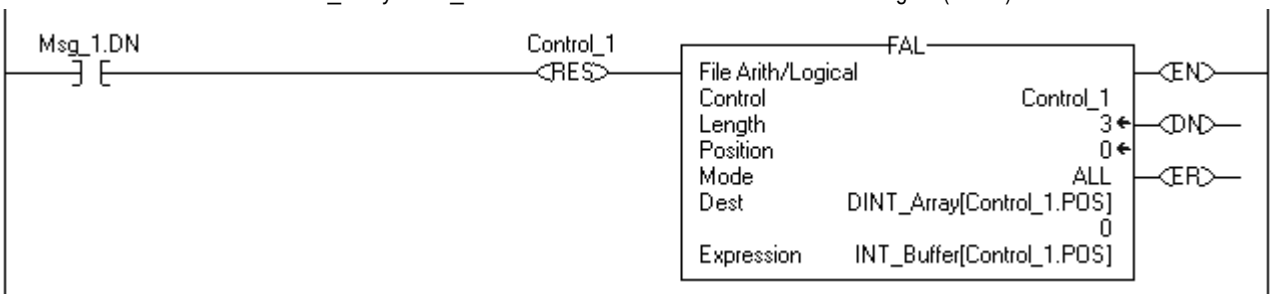
Read 3 integers from the PLC-5 controller and store them in INT_Buffer (3 INTs)



If Msg_1.DN =1 (MSG instruction has read the data.) then

Reset the FAL instruction.

The FAL instruction sets DINT_Array = INT_Buffer. This converts the values to 32-bit integers (DINTs).

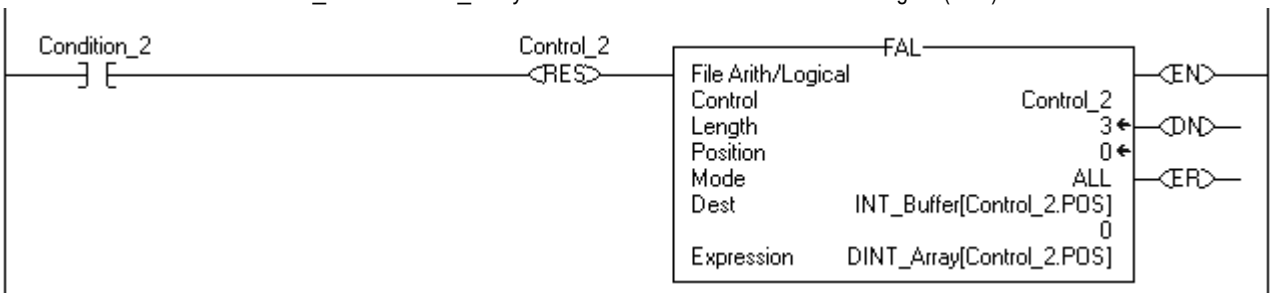


EXAMPLE Write integer values to a PLC-5 controller

If Condition_2 = 1 then:

Reset the FAL instruction.

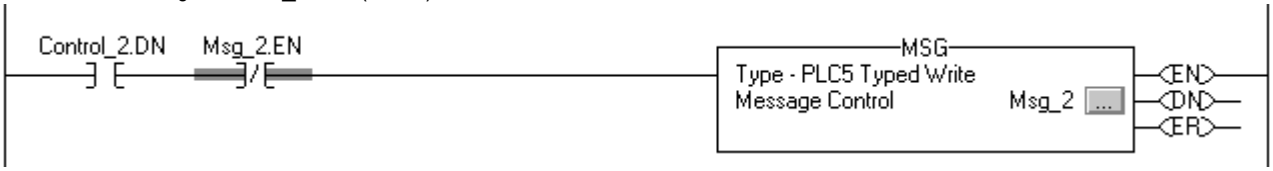
The FAL instruction sets INT_Buffer = DINT_Array. This converts the values to 16-bit integers (INTs).



If Control_2.DN = 1 (FAL instruction has converted the DINTs to INTs)

And Msg_2.EN = 0 (MSG instruction is not already enabled) then:

Write the integers in INT_Buffer (3 INTs) to the PLC-5 controller



Notes:

Producing a Large Array

When to Use This Chapter

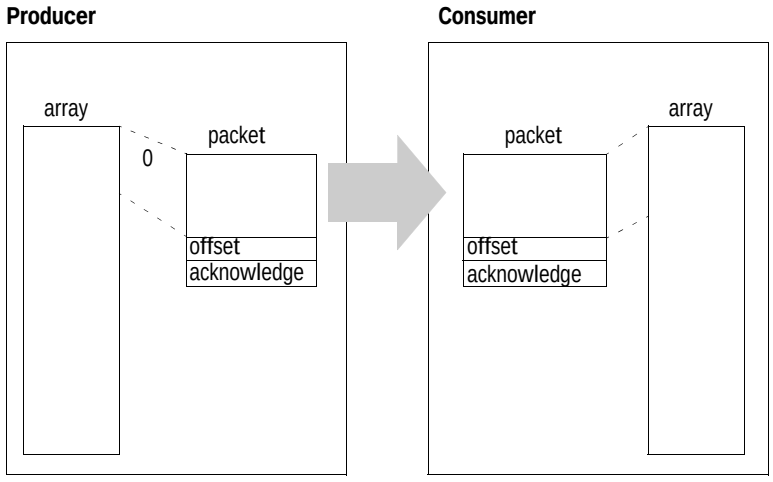
The Logix5000 controller can send as many as 500 bytes of data over a single scheduled connection. This corresponds to 125 DINT or REAL elements of an array. To transfer an array of more than 125 DINTs or REALs, use a produced/consumed tag of 125 elements to create a packet of data. You can then use the packet to send the array piecemeal to another controller.

When you send a large array of data in smaller packets, you must ensure that the transmission of a packet is complete before the data is moved into the destination array, for these reasons.

- Produced data over the ControlLogix backplane is sent in 50 byte segments.
- Data transmission occurs asynchronous to program scan.

The logic that this section includes uses an acknowledge word to make sure that each packet contains new data before the data moves to the destination array. The logic also uses an offset value to indicate the starting element of the packet within the array.

Because of the offset and acknowledge elements, each packet carries 123 elements of data from the array, as depicted below:



In addition, the array must contain an extra 122 elements. In other words, it must be 122 elements greater than the greatest number of elements that you want to transfer:

- These elements serve as a buffer.
- Since each packet contains the same number of elements, the buffer prevents the controller from copying beyond the boundaries of the array.
- Without the buffer, this would occur if the last packet contained fewer than 123 elements of actual data.

Produce a Large Array

1. In the Controller Tags folder of the controller project that produces the array, create these tags:

P	Tag Name	Type
	array_ack	DINT[2]
3	array_packet	DINT[125]

2. Convert array_ack to a consumed tag:.

For:	Specify:
Controller	name of the controller that is receiving the packet
Remote Tag Name	array_ack Both controllers use the same name for this data.

3. In either the Controller Tags folder or the tags folder of the program that will contain the logic for the transfer, create these tags:

Tag Name	Type
array	DINT[x] where x equals the number of elements to transfer plus 122 elements
array_offset	DINT
array_size	DINT
array_transfer_time	DINT
array_transfer_time_max	DINT
array_transfer_timer	TIMER

4. In the array_size tag, enter the number of elements of real data. (The value of x from step 3. minus the 122 elements of buffer.)

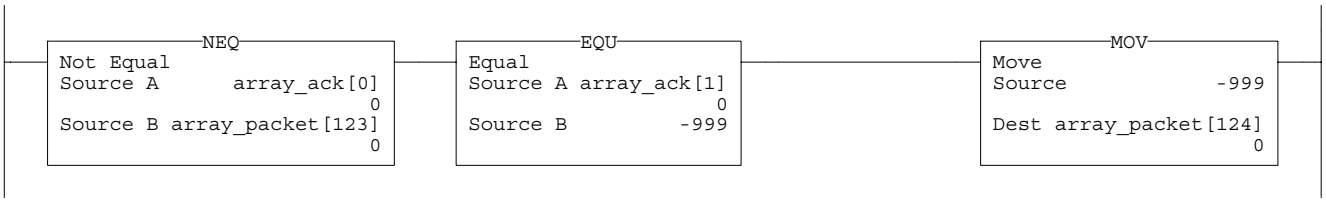
5. Create or open a routine for the logic that will create packets of data.

6. Enter this logic:

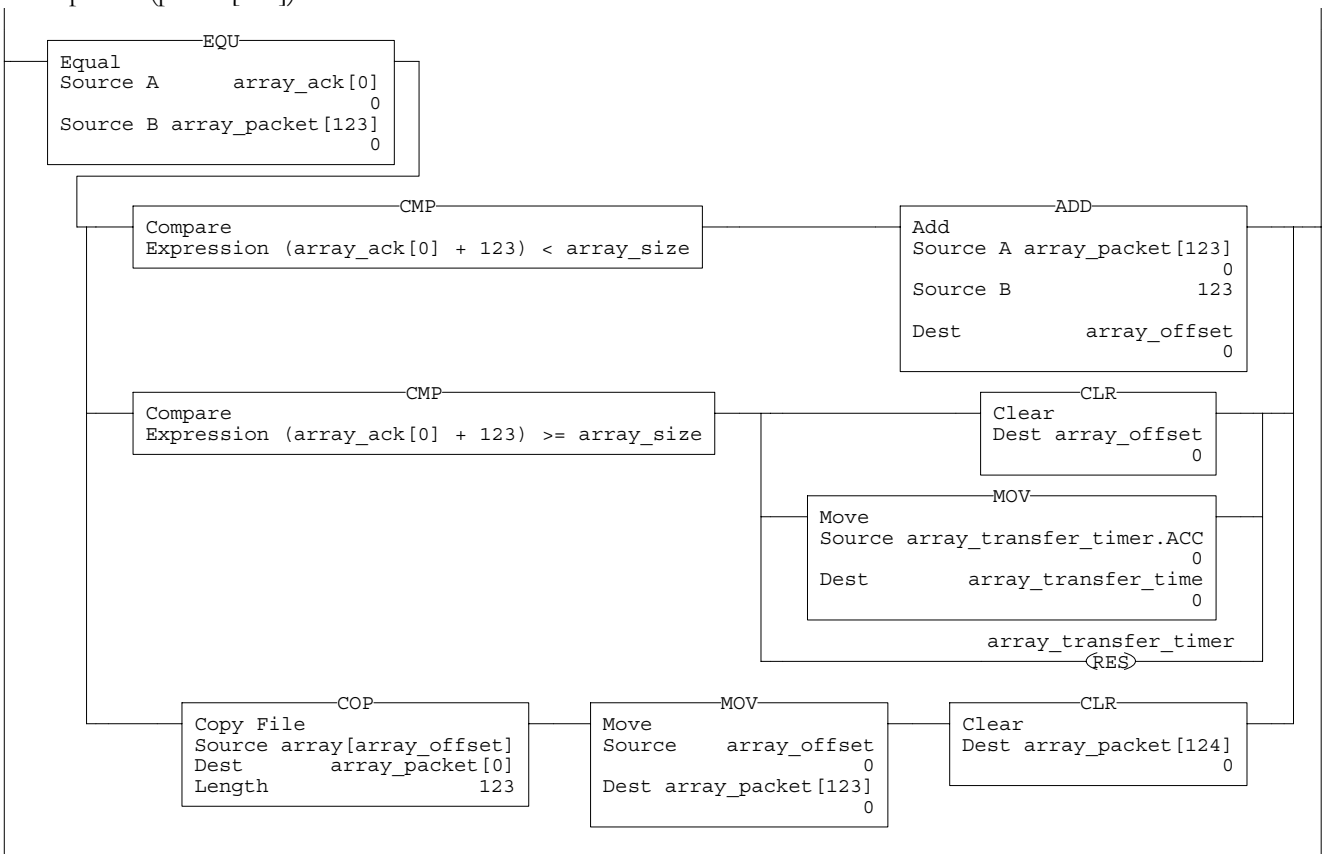
Times how long it takes to send the entire array



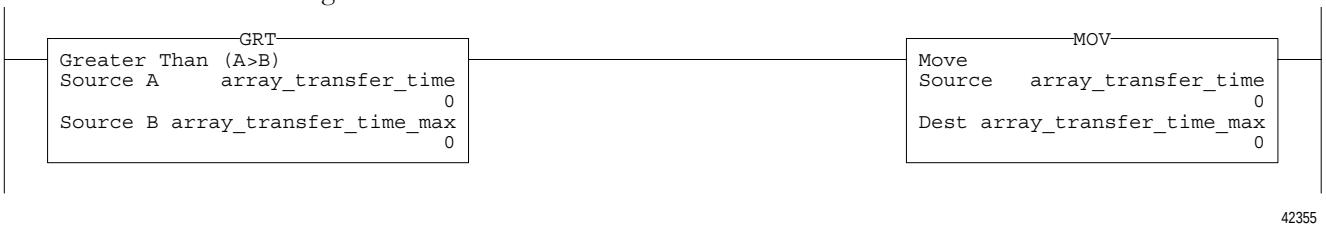
When the offset value in array_ack[0] is not equal to the current offset value but array_ack[1] equals -999, the consumer has begun to receive a new packet, so the rung moves -999 into the last element of the packet. The consumer waits until it receives the value -999 before it copies the packet to the array. This guarantees that the consumer has new data.



When the offset value in array_ack[0] is equal to the current offset value, the consumer has copied the packet to the array; so the rung checks for more data to transfer. If the offset value plus 123 is less than the size of the array, there is more data to transfer; so the rung increases the offset by 123. Otherwise, there is no more data to transfer; so the rung resets the offset value, logs the transfer time, and resets the timer. In either case, the rung uses the new offset value to create a new packet of data, appends the new offset value to the packet, and clears the acknowledge element of the packet (packet[124]).



If the current transfer time is greater than the maximum transfer time, updates the maximum transfer time. This maintains a record of the longest time to transfer data.



7. In the Controller Tags folder of the controller project that consumes the array, create these tags:

P	Tag Name	Type
3	array_ack	DINT[2]
	array_packet	DINT[125]

8. Convert array_packet to a consumed tag:

For	Specify
Controller	name of the controller that is sending the packet
Remote Tag Name	array_packet Both controllers use the same name for this data.

9. In either the Controller Tags folder or the tags folder of the program that will contain the logic for the transfer, create these tags:

Tag Name	Type
array	DINT[x] where x equals the number of elements to transfer plus 122 elements
array_offset	DINT

10. Create or open a routine for the logic that will move the data from the packets to the destination array.

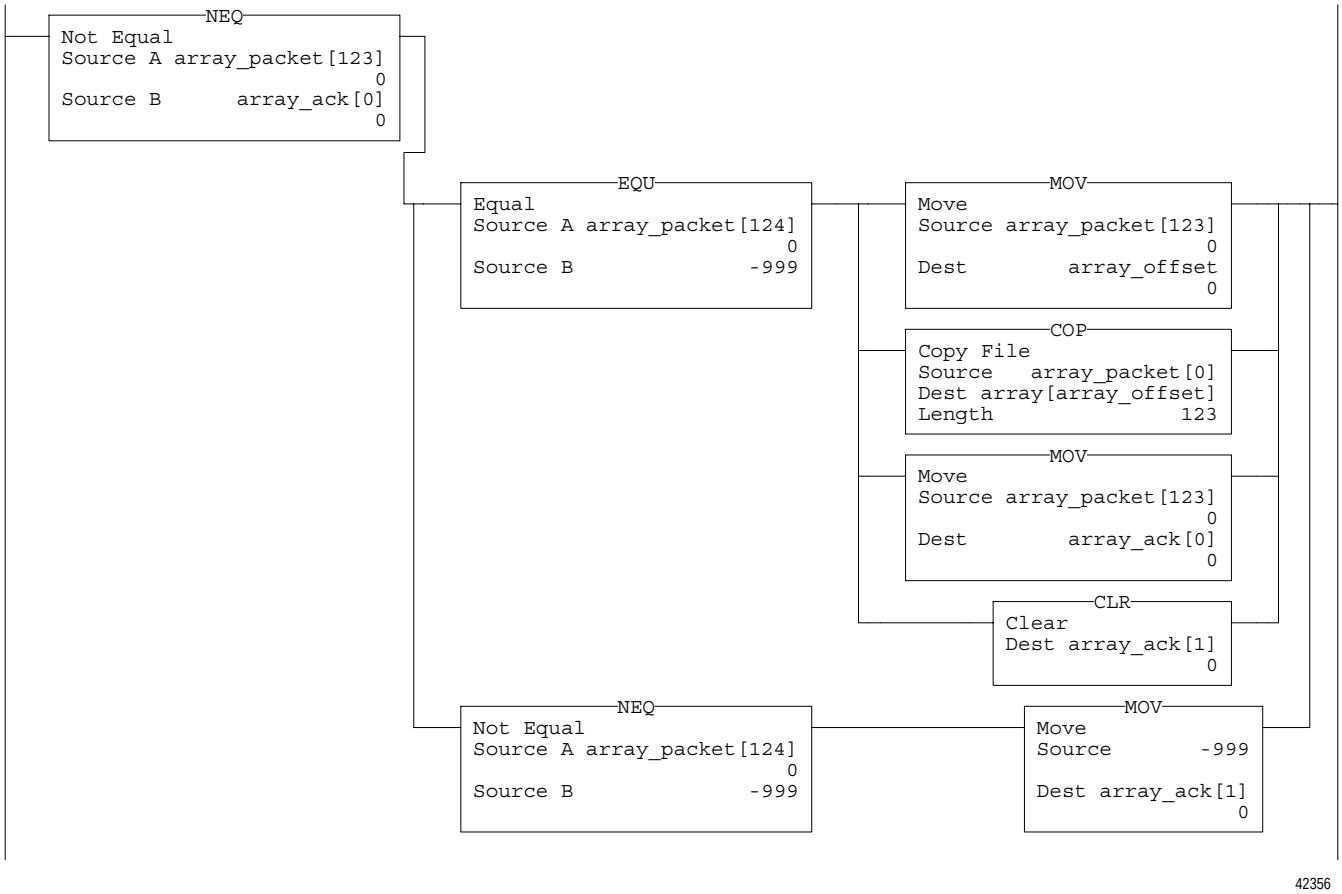
11. Enter this logic:

When the offset value in array_packet[123] is different than the offset value in array_ack[0], the controller has begun to receive a new packet of data; so the rung checks for the value of -999 in the last element of the packet.

If the last element of the packet equals -999, the controller has received an entire packet of new data and begins the copy operation:

- The offset value moves from the packet to array_offset.
- The COP instructions copies the data from the packet to the destination array, starting at the offset value.
- The offset value moves to array_ack[0], which signals that the copy is complete.
- Array_ack[1] resets to zero and waits to signal the arrival of a new packet.

If the last element of the packet is not equal -999, the transfer of the packet to the controller may not be complete; so -999 moves to array_ack[1]. This signals the producer to return the value of -999 in the last element of the packet to verify the transmission of the packet.



Transferring a large array as smaller packets improves system performance over other methods of transferring the data:

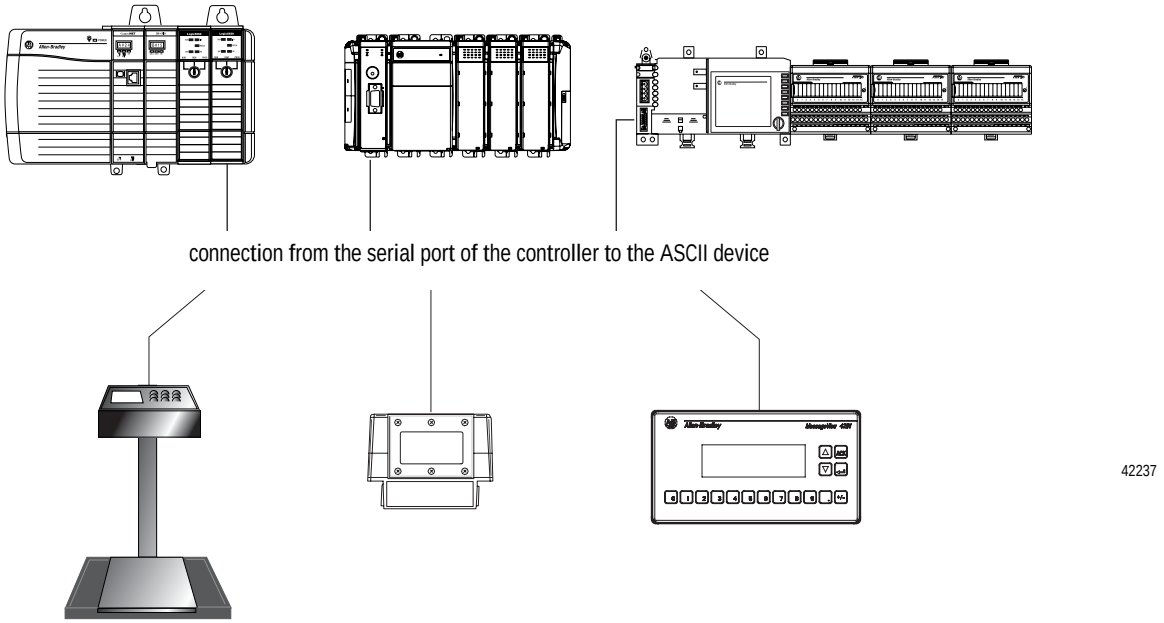
- Fewer connections are used than if you broke the data into multiple arrays and sent each as a produced tag. For example, an array with 5000 elements would take 40 connections (5000/125=40) using individual arrays.
- Faster transmission times are achieved than if you used a message instruction to send the entire array.
  - Messages are unscheduled and are executed only during the "system overhead" portion of the Logix5550 execution. Therefore, messages can take a fairly long time to complete the data transfer.
  - You can improve the transfer time by increasing system overhead time slice, but this diminishes the performance of the continuous task.

Communicating with an ASCII Device

When to Use This Chapter

Use this procedure to exchange ASCII data with a device through the serial port of the controller. For example, you can use the serial port to:

- read ASCII characters from a weigh scale module or bar code reader
- send and receive messages from an ASCII triggered device, such as a MessageView terminal.

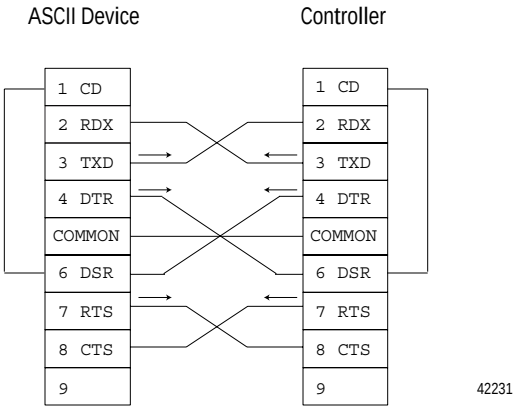
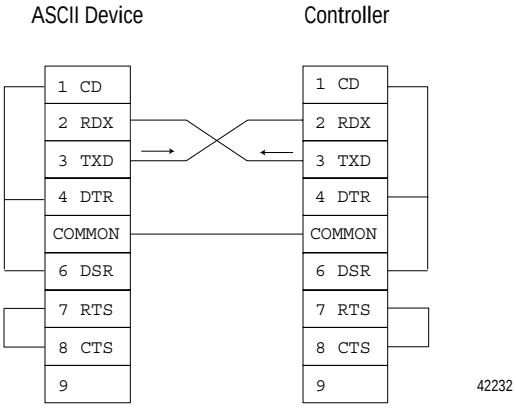


Topic	Page
Connect the ASCII Device	13-366
Configure the Serial Port	13-367
Configure the User Protocol	13-368
Create String Data Types	13-370
Read Characters from the Device	13-371
Send Characters to the Device	13-373
Enter ASCII Characters	13-375

In addition to the controller serial port, firmware revision 3.1 and greater of the 1756-EWEB EtherNet/IP Web Server module supports a socket interface that lets Logix5000 controllers exchange ASCII data using TCP or UDP socket services. See the EtherNet/IP Web Server User Manual, publication ENET-UM0527, revision C or later.

Connect the ASCII Device

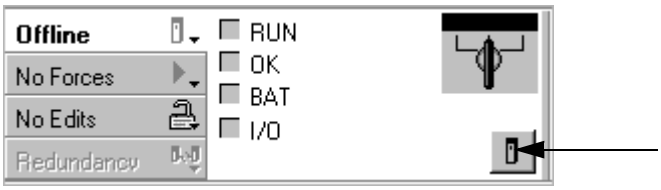
1. For the serial port of the ASCII device, determine which pins send signals and which pins receive signals.
2. Connect sending pins to corresponding receiving pins and attach jumpers:

If the communications:	Then wire the connectors:
handshake	
do not handshake	

3. Attach the cable shield to both connectors.
4. Connect the cable to the controller and the ASCII device.

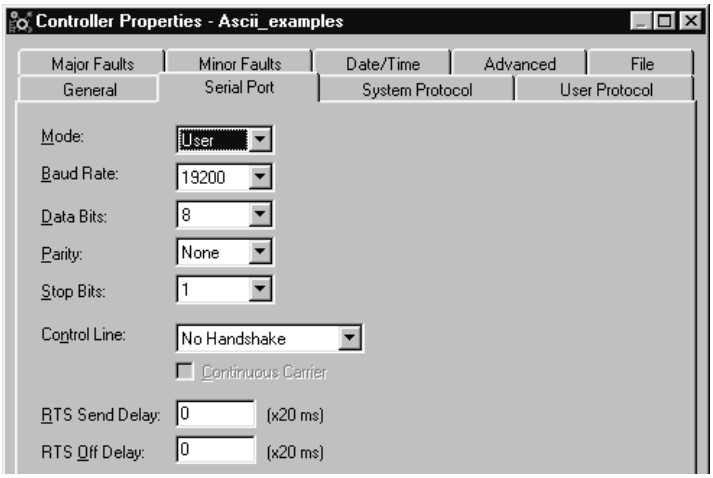
Configure the Serial Port

1. On the Online toolbar in the controller project, click the controller button.



2. Select the Serial Port tab.

3. Select User mode and enter the configuration settings for the serial port.



- Select the baud rate, data bits, parity, and stop bits.
- Select the Control Line option:

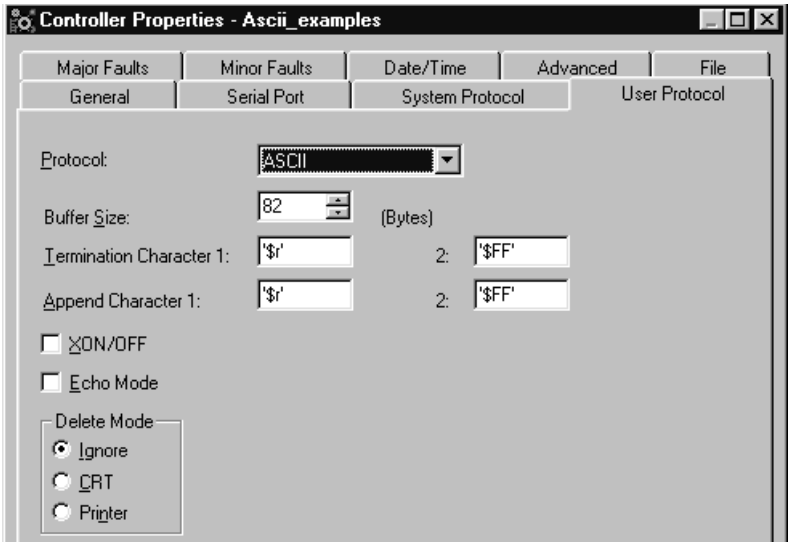
If	And	And this is the	Select	Then
you are **not** using a modem	→		No Handshaking	
you are using a modem	both modems in a point-to-point link are full-duplex	→	Full Duplex	
	master modem is full-duplex while slave modem is half-duplex	master controller.	Full Duplex	
		slave controller	Half Duplex	Check the Continuous Carrier check box.
	all modems in the system are half-duplex	→	Half Duplex	Clear the Continuous Carrier check box (default).

- For RTS Send Delay, enter the delay (in 20 ms units) between the time the RTS signal turns on (high) and the time that data is sent. For example, a value of 4 produces an 80 ms delay.
- For RTS Off Delay, enter the delay (in 20 ms units) between the time the last character is sent and the time that the RTS signal turns off (low).

4. Click Apply.

Configure the User Protocol

1. Select the User Protocol tab.



42252

- Enter a buffer size that is greater than or equal to the greatest number of characters in a transmission. (Twice the number of characters is a good guideline.)
- For ABL or ARL instructions, enter termination characters to mark the end of the data. For ASCII codes, see the back cover of this manual.

If the device sends	Then	Notes
one termination character	A. In the Termination Character 1 text box, type the hexadecimal ASCII code for the first character. B. In the Termination Character 2 text box, type $FF.	For printable characters, such as 1 or A, type the character.
two termination characters	In the Termination Character 1 and 2 text boxes, type the hexadecimal ASCII code for each character.	

- For AWA instruction, enter append characters. For ASCII codes, see the back cover of this manual.

To append	Then	Notes
one character	A. In the Append Character 1 text box, type the hexadecimal ASCII code for the first character. B. In the Append Character 2 text box, type $FF.	For printable characters, such as 1 or A, type the character.
two characters	In the Append Character 1 and 2 text boxes, type the hexadecimal ASCII code for each character.	

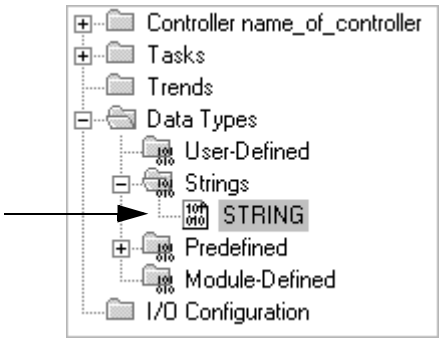
- If the ASCII device is configured for XON/XOFF flow control, select the XON/XOFF check box.
- If the ASCII device is a CRT or is pre-configured for half duplex transmission, select the Echo Mode check box.
- Select the Delete Mode:

If the ASCII device is	Select	Notes
CRT	CRT	• The DEL character ($7F) and the character that precedes the DEL character are **not** sent to the destination. • If echo mode is selected and an ASCII instruction reads the DEL character, the echo returns three characters: BACKSPACE SPACE BACKSPACE ($08 $20 $08).
printer	Printer	• The DEL character ($7F) and the character that precedes the DEL character are **not** sent to the destination. • If echo mode is selected and an ASCII instruction reads the DEL character, the echo returns two characters: / ($2F) followed by the character that was deleted.
None of the above	Ignore	The DEL character ($7F) is treated as any other character.

2. Click OK.

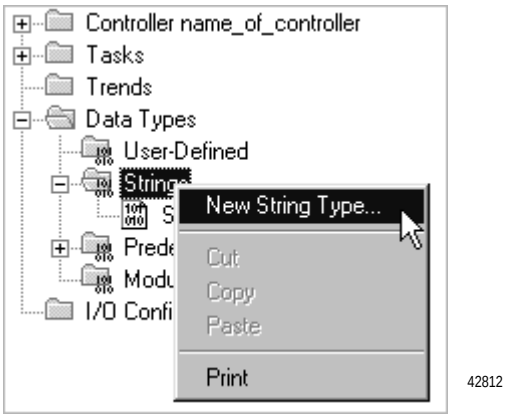
Create String Data Types

Store ASCII characters in tags that use a **string** data type.



You can use the default STRING data type. It stores up to 82 characters.

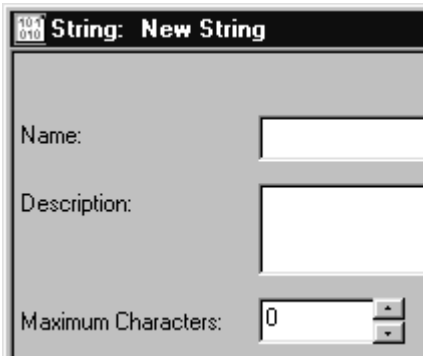
or



You can create a new string data type to store the number of characters that you define.

IMPORTANT Use caution when you create a new string data type. If you later decide to change the size of the string data type, you may lose data in any tags that currently use that data type.

If you	Then
make a string data type smaller	• The data is truncated. • The LEN is unchanged.
make a string data type larger	The data and LEN is reset to zero.



1. In the controller organizer, right-click Strings and choose New String Type…
2. Type a name for the data type.
3. Type the maximum number characters that this string data type will store.
4. Click OK.

Read Characters from the Device

As a general rule, before you read the buffer use an ACB or ABL instruction to verify that the buffer contains the required characters:

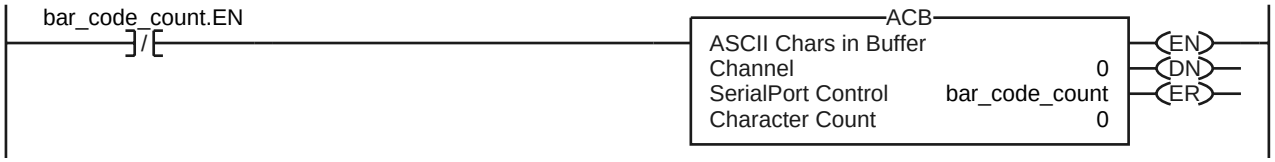
- An ARD or ARL instruction continues to read the buffer until the instruction reads the required characters.
- While an ARD or ARL instruction is reading the buffer, no other ASCII Serial Port instructions, except the ACL, can execute.
- Verifying that the buffer contains the required characters prevents the ARD or ARL from holding up the execution of other ASCII Serial Port instructions while the input device sends its data.

For additional information on ASCII Serial Port instructions, see *Logix5000 Controllers General Instruction Set Reference Manual*, publication 1756-RM003.

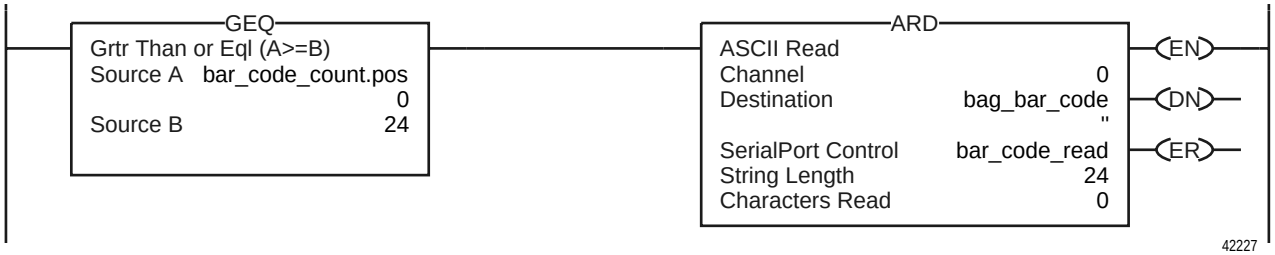
For example, if the device sends s fixed number of characters, such as a bar code reader:

EXAMPLE

A bar code reader sends bar codes to the serial port (channel 0) of the controller. Each bar code contains 24 characters. To determine when the controller receives a bar code, the ACB instruction continuously counts the characters in the buffer.



When the buffer contains at least 24 characters, the controller has received a bar code. The ARD instruction moves the bar code to the bag_bar_code tag.



For example, if the device sends a variable number of characters, such as a message or display terminal:

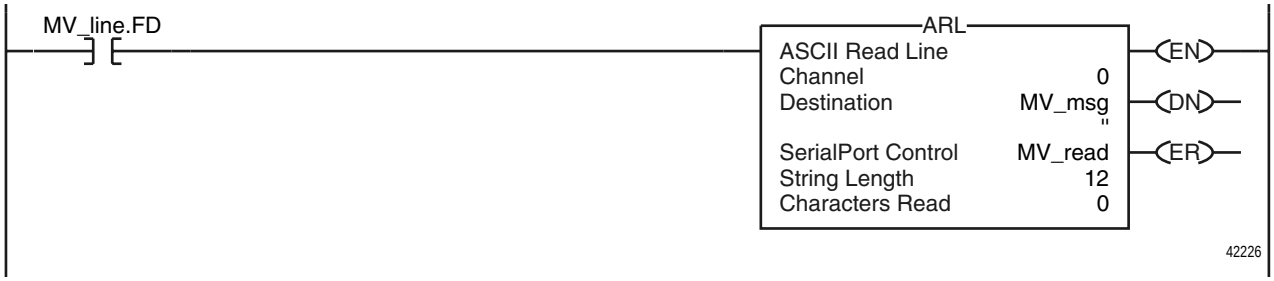
EXAMPLE

Continuously tests the buffer for a message.

- Since each message ends in a carriage return ($0D), the carriage return is configured as the termination character in the Controller Properties dialog box, User Protocol tab.
- When the ABL finds a carriage return, its sets the FD bit.



When the ABL instruction finds the carriage return (MV_line.FD is set), the controller removes the characters from the buffer, up to and including the carriage return, and places them in the MV_msg tag.



Send Characters to the Device

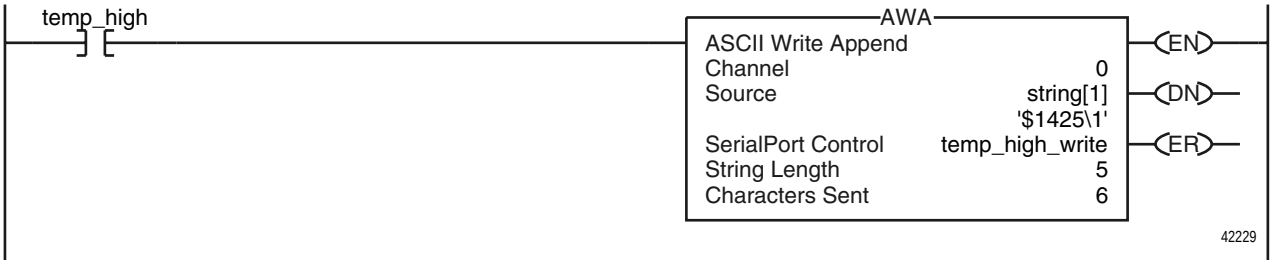
When you send characters to the device, you need to determine whether you will always send the same number of characters each time and whether you want to append terminations characters to the data.

For example, if you always send the same number of characters and want to automatically append one or two characters to the end of the data:

EXAMPLE

When the temperature exceeds the high limit (temp_high is on), the AWA instruction sends five characters from the string[1] tag to a MessageView terminal.

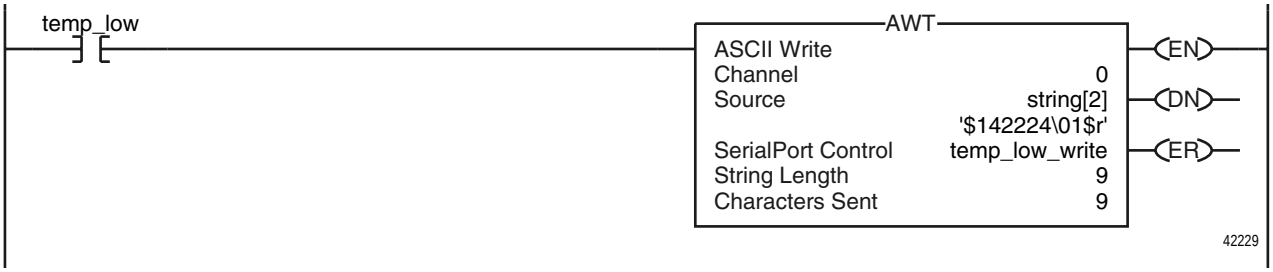
- The $14 counts as one character. It is the hex code for the Ctrl-T character.
- The instruction also sends (appends) the characters defined in the user protocol. In this example, the AWA instruction sends a carriage return ($0D), which marks the end of the message.



And then to always send the same number of characters:

EXAMPLE

When the temperature reaches the low limit (temp_low is on), the AWT instruction sends nine characters from the string[2] tag to a MessageView terminal. (The $14 counts as one character. It is the hex code for the Ctrl-T character.)

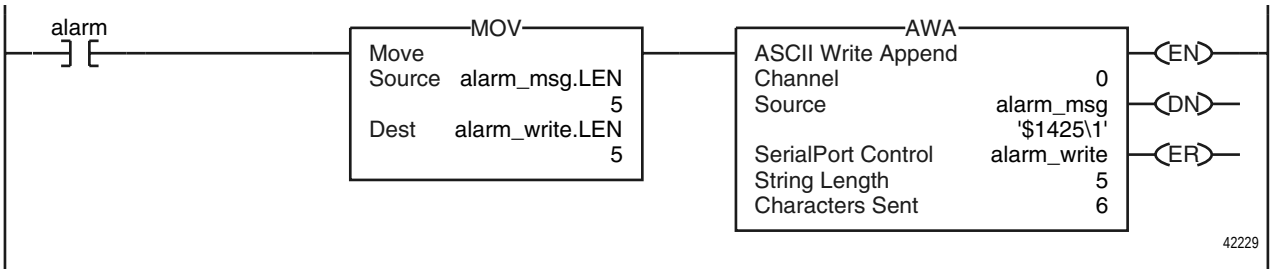


For example, if you send a different number of characters each time and want to automatically append one or two characters to the end of the data:

EXAMPLE

When alarm is on, the AWA instruction sends the characters in alarm_msg and appends a termination character.

- Because the number of characters in alarm_msg varies, the rung first moves the length of alarm_msg (alarm_msg.LEN) to the length of the AWA instruction (alarm_write.LEN).
- In alarm_msg, the $14 counts as one character. It is the hex code for the Ctrl-T character.

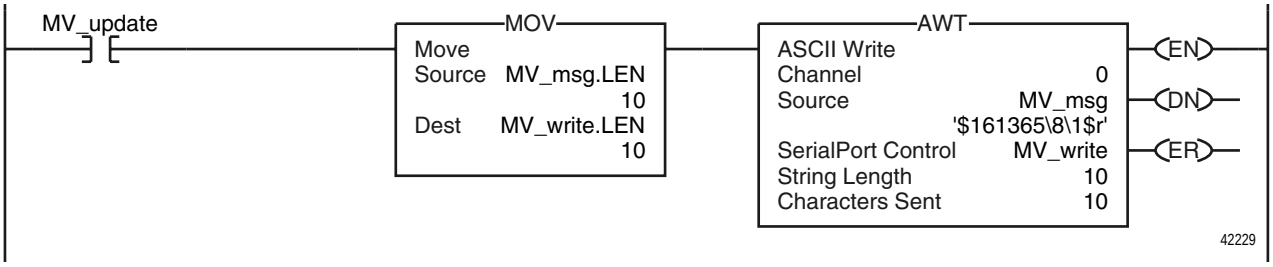


And then to send a different number of characters each time:

EXAMPLE

When MV_update is on, the AWT instruction sends the characters in MV_msg.

- Because the number of characters in MV_msg varies, the rung first moves the length of MV_msg (MV_msg.LEN) to the length of the AWT instruction (MV_write.LEN).
- In MV_msg, the $16 counts as one character. It is the hex code for the Ctrl-V character.

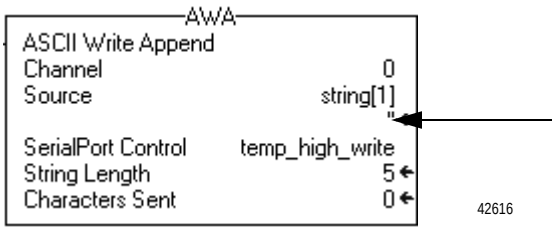


Enter ASCII Characters

If	Then
You want logic to create the string.	Go to "Processing ASCII Characters" on page 14-377.
You want to enter the characters.	Go to step 1.

IMPORTANT This String Browser window shows the characters up to the value of the LEN member of the string tag. The string tag may contain additional data, which the String Browser window does not show.

1. Double-click the value area of the Source.



A text entry box appears:



2. Enter the characters for the string.

3. Click OK.

Notes:

Processing ASCII Characters

When to Use This Chapter

Use this chapter to:

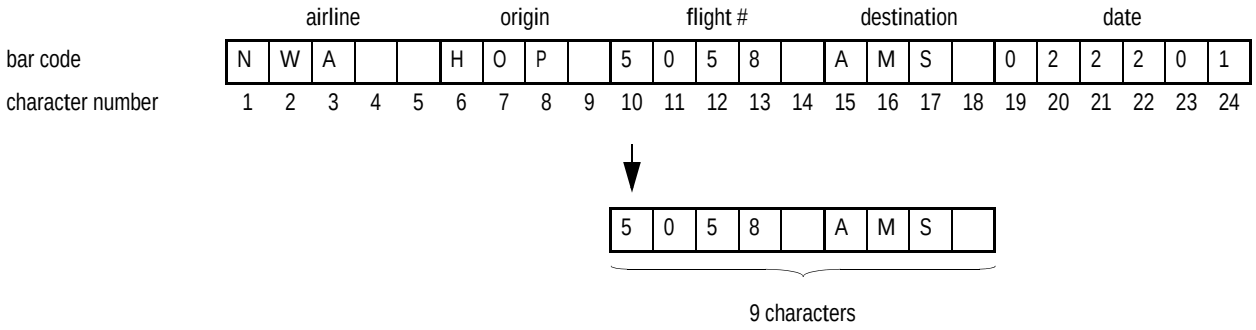
- interpret a bar code and take action based on the bar code
- use a weight from a weigh scale when the weight is sent as ASCII characters
- decode a message from an ASCII triggered device, such as an operator terminal
- build a string for an ASCII triggered device using variables from your application

Topic	Page
Extract a Part of a Bar Code	378
Look Up a Bar Code	379
Check the Bar Code Characters	382
Convert a Value	383
Decode an ASCII Message	384
Build a String	385

For additional information on ASCII-related instructions, refer to *Logix5000 Controllers General Instruction Set Reference Manual*, publication 1756-RM003.

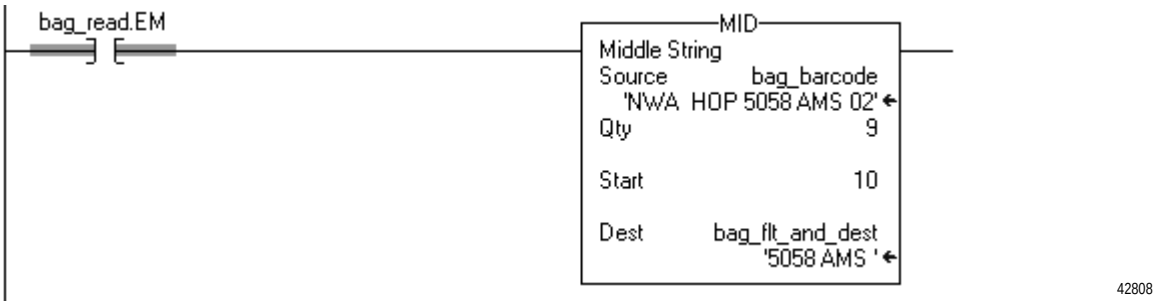
Extract a Part of a Bar Code

For example, a bar code may contain information about a bag on a conveyor at an airport. To check the flight number and destination of the bag, you extract characters 10 - 18.



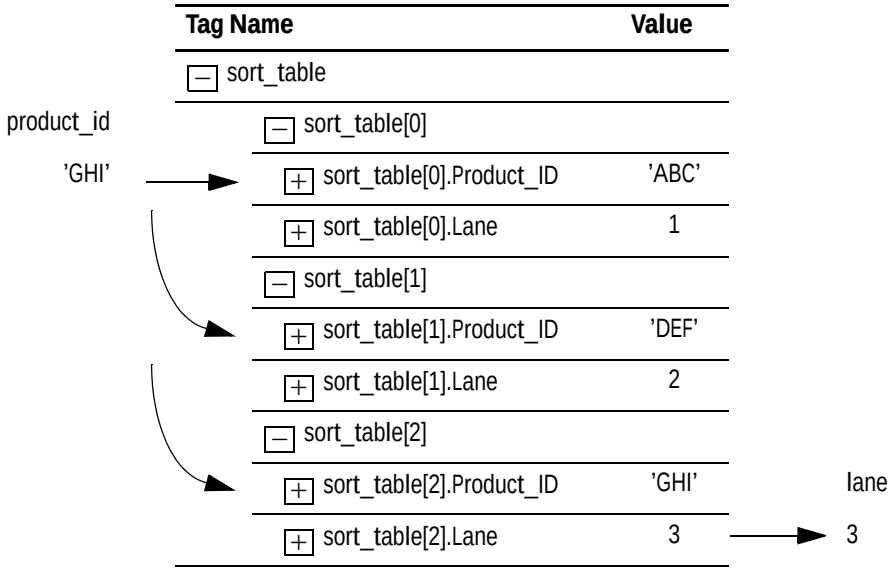
EXAMPLE

In the baggage handling conveyor of an airport, each bag gets a bar code. Characters 10 - 18 of the bar code are the flight number and destination airport of the bag. After the bar code is read (bag_read.EM is on) the MID instruction copies the flight number and destination airport to the bag_flt_and_dest tag.



Look Up a Bar Code

For example, in a sorting operation, an array of a user-defined data type creates a table that shows the lane number for each type of product. To determine which lane to route a product, the controller searches the table for the product ID (characters of the bar code that identify the product).

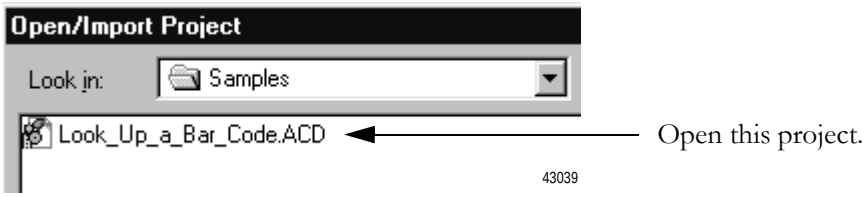


To look up a bar code:

- Create the PRODUCT_INFO Data Type
- Search for the Characters
- Identify the Lane Number
- Reject Bad Characters
- Enter the Product IDs and Lane Numbers

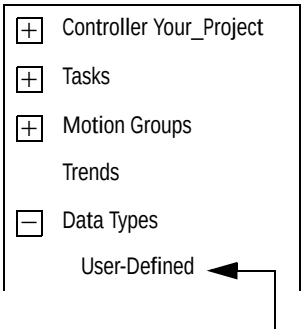
TIP

To copy the above components from a sample project, open the …\RSLogix 5000\Projects\Samples folder.



Create the PRODUCT_INFO Data Type

To create a new data type:

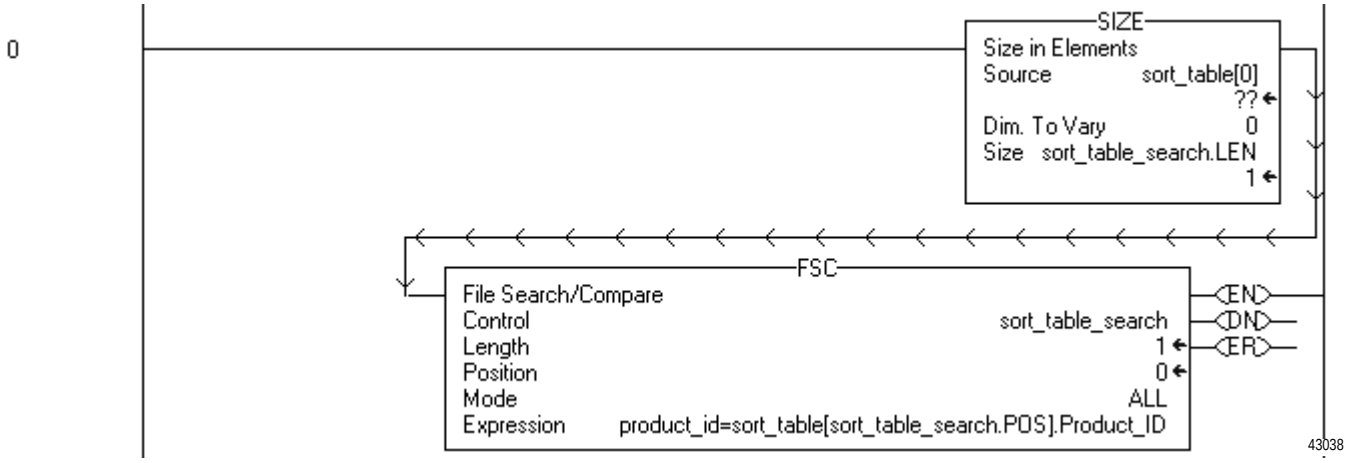


Right-click and choose *New Data Type*.

Create this user-defined data type.

Data Type: PRODUCT_INFO				
Name	PRODUCT_INFO			
Description	Identifies the destination for an item based on an ASCII string of characters that identify the item			
Members				
	Name	**Data Type**	**Style**	**Description**
	Product_ID	STRING		ASCII characters that identify the item
	Lane	DINT	Decimal	Destination for the item, based on its ID

Search for the Characters



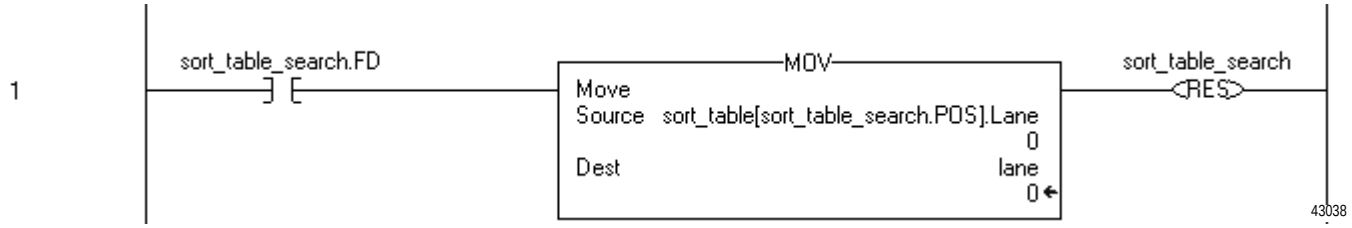
The SIZE instruction:

- counts the number of elements in the sort_table array (type PRODUCT_INFO). This array contains the product ID for each item and the corresponding lane number for the item.
- counts the number of elements in Dimension 0 of the array. In this case, that is the only dimension.
- sets the Length of the subsequent FSC instruction equal to the size of the sort_table array.

The FSC instruction searches each Product_ID member in the sort_table array until the instruction finds a match to the product_id tag.

- the sort_table_search tag controls the FSC instruction
- although the previous instruction sets the Length of this instruction, you enter an initial value to verify the project
- the product_id tag contains the bar code characters that you want to find

Identify the Lane Number



When the FSC instruction finds the product ID within the sort_table array, the instruction sets the FD bit. The POS member indicates the element number within the sort_table array of the match. The corresponding LANE member indicates the lane number of the match.

Based on the POS value, the MOV instruction moves the corresponding lane number into the lane tag. The controller uses the value of this tag to route the item.

After the MOV instruction sets the value of the lane tag, the RES instruction resets the FSC instruction so it can search for the next product ID.

Reject Bad Characters



If the FSC instruction does not find the product ID within the sort_table array, the instruction sets the DN bit. The MOV instruction moves 999 into the lane tag to notify the controller to reject or reroute the item.

After the MOV instruction sets the value of the lane tag, the RES instruction resets the FSC instruction so it can search for the next product ID.

Enter the Product IDs and Lane Numbers

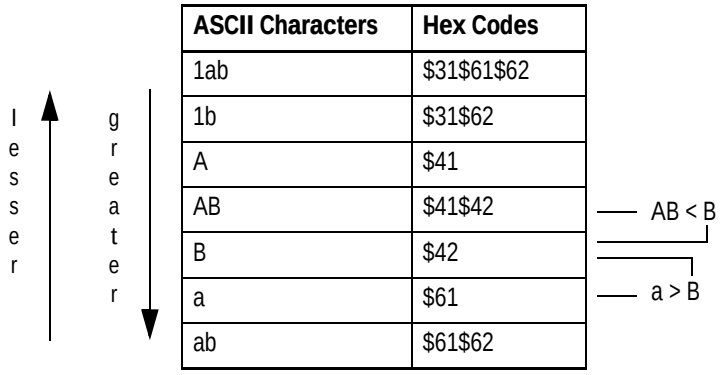
In the sort_table array, enter the ASCII characters to identify each item and the corresponding lane number for the item.

Tag Name	Value
− sort_table	{…}
− sort_table[0]	{…}
+ sort_table[0].Product_ID	ASCII characters that identify the first item
+ sort_table[0].Lane	lane number for the item
− sort_table[1]	{…}
+ sort_table[1].Product_ID	ASCII characters that identify the next item
+ sort_table[1].Lane	lane number for the item

Check the Bar Code Characters

Use a compare instruction (EQU, GEQ, GRT, LEQ, LES, NEQ) to check for specific characters.

- The hexadecimal values of the characters determine if one string is less than or greater than another string.
- When the two strings are sorted, as in a telephone directory, the order of the strings determines which one is greater.



Use one of these compare instruction:

To see if the string is:	Enter this instruction:
equal to specific characters	EQU
not equal to specific characters	NEQ
greater than specific characters	GRT
equal to or greater than specific characters	GEQ
less than specific characters	LES
equal to or less than specific characters	LEQ

For example:

EXAMPLE

When bag_flt_and_dest is equal to gate[1], xfer[1] turns on. This routes the bag to the required gate.



Convert a Value

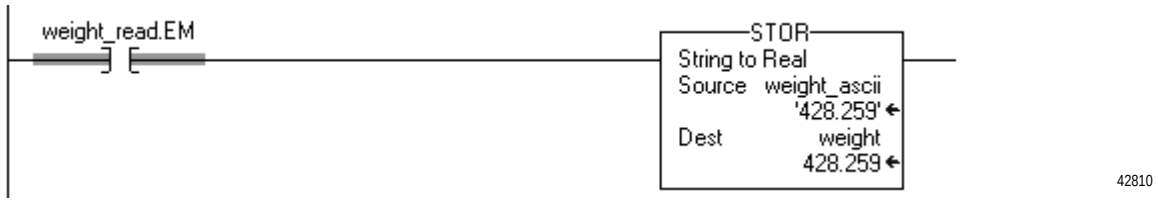
You can convert the ASCII representation of a value to an DINT or REAL value that you can use in your application.

- The STOD and STOR instructions skip any initial control or non-numeric characters (except the minus sign in front of a number).
- If the string contains multiple groups of numbers that are separated by delimiters (e.g., /), the STOD and STOR instructions convert only the first group of numbers.

For example, to convert ASCII characters to a floating-point value:

EXAMPLE

After reading the weight from the scale (weight_read.EM is on), the STOR instruction converts the numeric characters in weight_ascii to a REAL value and stores the result in weight.



For example, to convert ASCII characters to an integer value:

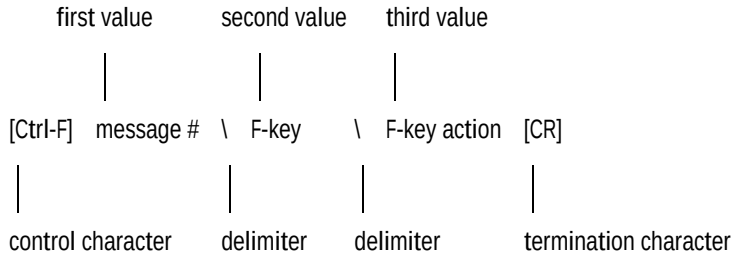
EXAMPLE

When MV_read.EM is on, the STOD instruction converts the first set of numeric characters in MV_msg to an integer value. The instruction skips the initial control character ($06) and stops at the delimiter (\ ).

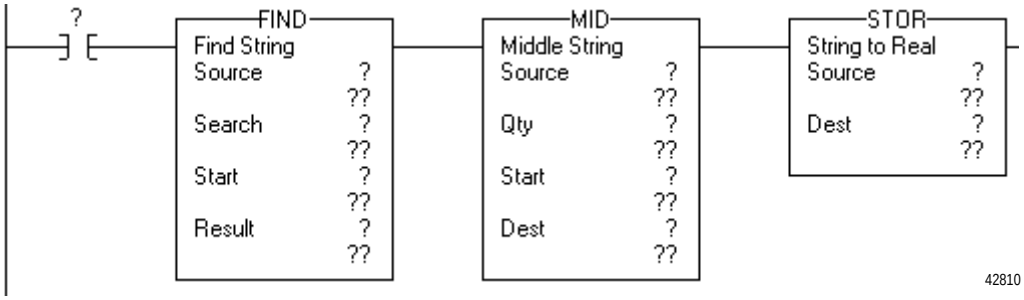


Decode an ASCII Message

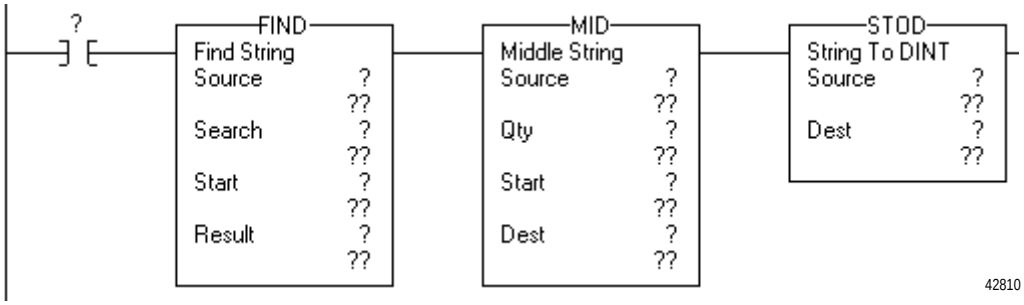
You can extract and convert a value from an ASCII message that contains multiple values. For example, a message may look like this:



Rung A: Find and Convert a Floating-Point Value



Rung B: Find and Convert an Integer Value



The FIND instruction locates characters within a string:

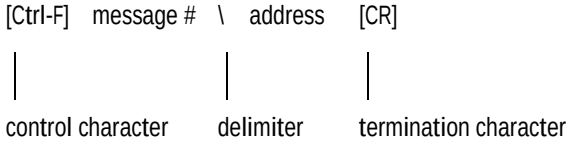
- The Source contains the string tag to search.
- The Result contains the location where the FIND instruction locates the search value you specify.

The MID instruction identifies a group of characters within a string and places them in their own string tag:

- The source is the same string tag as for the FIND instruction.
- The quantity values tells the MID instruction how many characters to pull from the source.
- The start value is the same as the Result value from the FIND instruction. This tells the MID instruction where to start pulling characters from the Source.
- The Destination contains the characters you located.

Build a String

This example builds a string that contains two variables. For example, an operator terminal may require a string that looks like this:



- For more variables, use additional INSERT or CONCAT instructions.
- If you need to send a floating-point value, use a RTOS instruction in place of the DTOS instruction.
- The final string does not include the termination character. When you send the string, use an AWA instruction to automatically append the termination character.

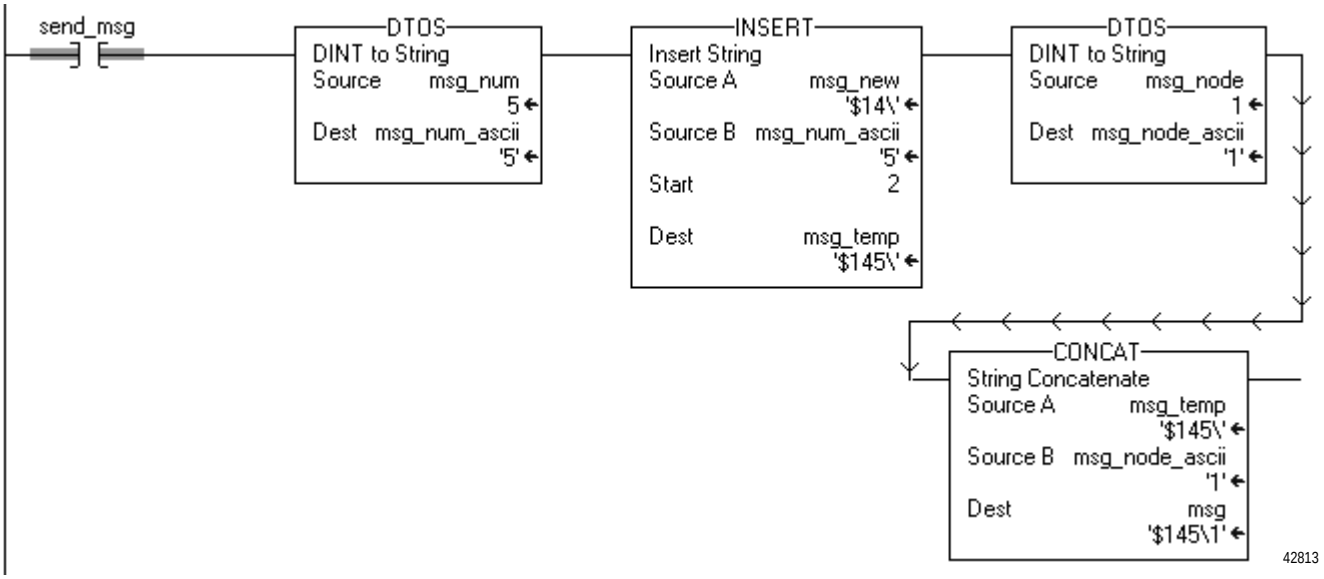
EXAMPLE

To trigger a message in a MessageView terminal, the controller sends the terminal a message in this format: [Ctrl-T] message # \ address [CR]

When send_msg is on, the rung does this:

- The first DTOS instruction converts the message number to ASCII characters.
- The INSERT instruction inserts the message number (in ASCII) after the control character [Ctrl-T]. (The hex code for Ctrl-T is $14.)
- The second DTOS instruction converts the node number of the terminal to ASCII characters.
- The CONCAT instruction puts the node number (in ASCII) after the backslash [\ ] and stores the final string in msg.

To send the message, an AWA instruction sends the msg tag and appends the carriage return [CR].



Forcing Logic Elements

When to Use This Chapter

Use a force to override data that your logic either uses or produces. For example, use forces in the following situations:

- test and debug your logic
- check wiring to an output device
- temporarily keep your process functioning when an input device has failed

Use forces only as a temporary measure. They are not intended to be a permanent part of your application.

Topic	Page
Precautions	388
Check Force Status	389
What to Force	391
When to Use an I/O Force	391
Add an I/O Force	392
When to Use Step Through	393
Step Through a Transition or a Force of a Path	393
When to Use an SFC Force	393
Add an SFC Force	395
Remove or Disable Forces	396

Precautions

When you use forces, take these precautions:

ATTENTION



Forcing can cause unexpected machine motion that could injure personnel. Before you use a force, determine how the force will effect your machine or process and keep personnel away from the machine area.

- Enabling I/O forces causes input, output, produced, or consumed values to change.
- Enabling SFC forces causes your machine or process to go to a different state or phase.
- Removing forces may still leave forces in the enabled state.
- If forces are enabled and you install a force, the new force immediately takes effect.

Enable Forces

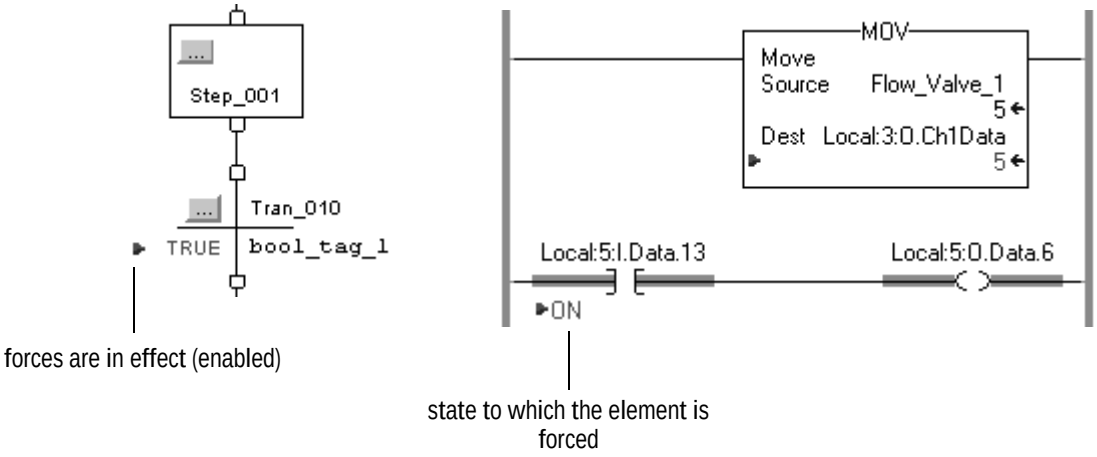
For a force to take effect, you enable forces. You can only enable and disable forces at the controller level.

- You can enable I/O forces and SFC forces separately or at the same time.
- You cannot enable or disable forces for a specific module, tag collection, or tag element.

IMPORTANT

If you download a project that has forces enabled, the programming software prompts you to enable or disable forces after the download completes.

When forces are in effect (enabled), a ↵ appears next to the forced element.



Disable or Remove a Force

To stop the effect of a force and let your project execute as programmed, disable or remove the force.

- You can disable or remove I/O and SFC forces at the same time or separately.
- Removing a force on an alias tag also removes the force on the base tag.

ATTENTION



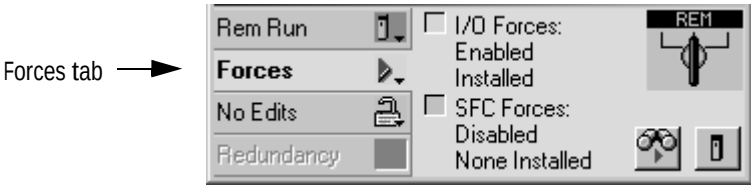
Changes to forces can cause unexpected machine motion that could injure personnel. Before you disable or remove forces, determine how the change will effect your machine or process and keep personnel away from the machine area.

Check Force Status

Before you use a force, determine the status of forces for the controller. You can check force status:

To determine the status of:	Use any of the following:
I/O forces	• online toolbar • FORCE LED • GSV instruction
SFC forces	Online Toolbar

The Online toolbar shows the status of forces. It shows the status of I/O forces and SFC forces separately.



This:	Means:
Enabled	• If the project contains any forces of this type, they *are* overriding your logic. • If you add a force of this type, the new force immediately takes effect
Disabled	Forces of this type are inactive. If the project contains any forces of this type, they *are not* overriding your logic.
Installed	At least one force of this type exists in the project.
None Installed	No forces of this type exist in the project.

FORCE LED

If your controller has a FORCE LED, use the LED to determine the status of any I/O forces.

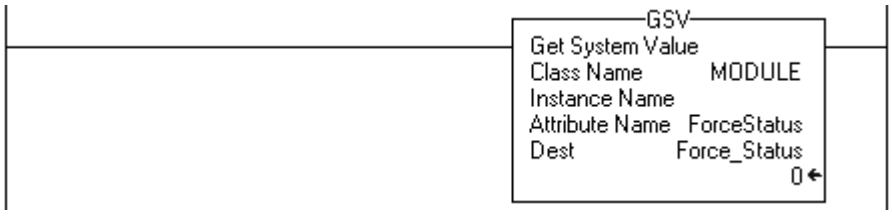
IMPORTANT The FORCE LED shows only the status of I/O forces. It does not show that status of SFC forces.

If the FORCE LED is:	Then:
off	• No tags contain force values. • I/O forces are inactive (disabled).
flashing	• At least one tag contains a force value. • I/O forces are inactive (disabled).
solid	• I/O forces are active (enabled). • Force values may or may not exist.

GSV Instruction

IMPORTANT The ForceStatus attribute shows only the status of I/O forces. It does not show the status of SFC forces.

This example shows how to use a GSV instruction to get the status of forces.



where:

Force_Status is a DINT tag.

To determine if:	Examine this bit:	For this value:
forces are installed	0	1
no forces are installed	0	0
forces are enabled	1	1
forces are disabled	1	0

What to Force

You can force these elements of a project:

If you want to:	Then:
override an input value, output value, produced tag, or consumed tag	Add an I/O Force
override the conditions of a transition one time to go from an active step to the next step	Step Through a Transition or a Force of a Path
override one time the force of a simultaneous path and execute the steps of the path	
override the conditions of a transition in a sequential function chart	Add an SFC Force
execute some but not all the paths of a simultaneous branch of a sequential function chart	

When to Use an I/O Force

Use an I/O force to:

- override an input value from another controller (i.e., a consumed tag)
- override an input value from an input device
- override your logic and specify an output value for another controller (i.e., a produced tag)
- override your logic and specify the state of an output device

IMPORTANT Forcing increases logic execution time. The more values you force, the longer it takes to execute the logic.

IMPORTANT I/O forces are held by the controller and not by the programming workstation. Forces remain even if the programming workstation is disconnected.

When you force an I/O value:

- You can force all I/O data, except for configuration data.
- If the tag is an array or structure, such as an I/O tag, force a BOOL, SINT, INT, DINT, or REAL element or member.
- If the data value is a SINT, INT, or DINT, you can force the entire value or you can force individual bits within the value. Individual bits can have a force status of:
  - no force
  - force on
  - force off
- You can also force an alias to an I/O structure member, produced tag, or consumed tag.
  - An alias tag shares the same data value as its base tag, so forcing an alias tag also forces the associated base tag.
  - Removing a force from an alias tag removes the force from the associated base tag.

Force an Input Value

Forcing an input or consumed tag:

- overrides the value regardless of the value of the physical device or produced tag
- does not affect the value received by other controllers monitoring that input or produced tag

Force an Output Value

Forcing an output or produced tag overrides the logic for the physical device or other controller (s). Other controllers monitoring that output module in a listen-only capacity will also see the forced value.

Add an I/O Force

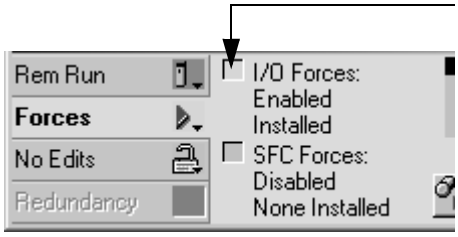
To override an input value, output value, produced tag, or consumed tag, use an I/O force:

ATTENTION



Forcing can cause unexpected machine motion that could injure personnel. Before you use a force, determine how the force will effect your machine or process and keep personnel away from the machine area.

- Enabling I/O forces causes input, output, produced, or consumed values to change.
- If forces are enabled and you install a force, the new force immediately takes effect.



1. What is the state of the I/O Forces indicator?

If:	Then note:
off	No I/O forces currently exist.
flashing	No I/O forces are active. But at least one force already exists in your project. When you enable I/O forces, *all* existing I/O forces will also take effect.
solid	I/O forces are enabled (active). When you install (add) a force, it immediately takes effect.

2. Open the routine that contains the tag that you want to force.
3. Right-click the tag and choose Monitor… If necessary, expand the tag to show the value that you want to force (e.g., BOOL value of a DINT tag).

4. Install the force value:

To force a:	Do this:
BOOL value	Right-click the tag and choose Force ON or Force OFF.
non-BOOL value	In the Force Mask column for the tag, type the value to which you want to force the tag. Then press the Enter key.

5. Are I/O forces enabled? (See step 1.)

If:	Then:
no	From the Logic menu, choose I/O Forcing > Enable All I/O Forces. Then choose Yes to confirm.
yes	Stop.

When to Use Step Through

To override a false transition one time and go from an active step to the next step, use the Step Through option. With the Step Through option:

- You do not have to add, enable, disable, or remove forces.
- The next time the SFC reaches the transition, it executes according to the conditions of the transition.

This option also lets you override one time the false force of a simultaneous path. When you step through the force, the SFC executes the steps of the path.

Step Through a Transition or a Force of a Path

To step through the transition of an active step or a force of a simultaneous path:

1. Open the SFC routine.
2. Right-click the transition or the path that is forced and choose Step Through.

When to Use an SFC Force

To override the logic of an SFC, you have these options:

If you want to:	Then:
override the conditions of a transition each time the SFC reaches the transition	Force a Transition
prevent the execution of one or more paths of a simultaneous branch	Force a Simultaneous Path

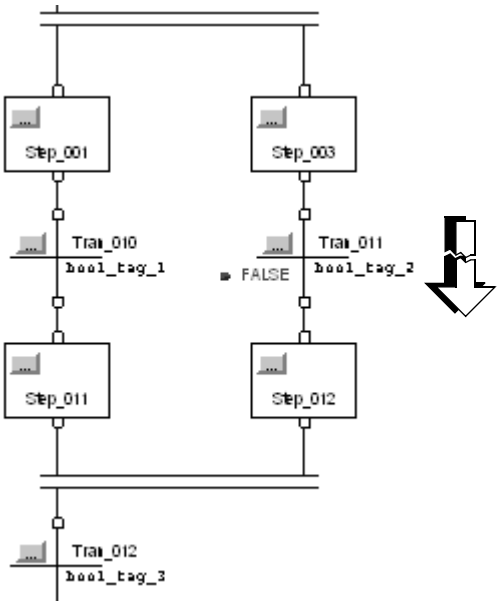
Force a Transition

To override the conditions of a transition through repeated executions of an SFC, force the transition. The force remains until you remove it or disable forces.

If you want to:	Then:
prevent the SFC from going to the next step	force the transition false
cause the SFC go to the next step regardless of transition conditions	force the transition true

If you force a transition within a simultaneous branch to be false, the SFC stays in the simultaneous branch as long as the force is active (installed and enabled).

- To leave a simultaneous branch, the last step of each path must execute at least one time and the transition below the branch must be true.
- Forcing a transition false prevents the SFC from reaching the last step of a path.
- When you remove or disable the force, the SFC can execute the rest of the steps in the path.

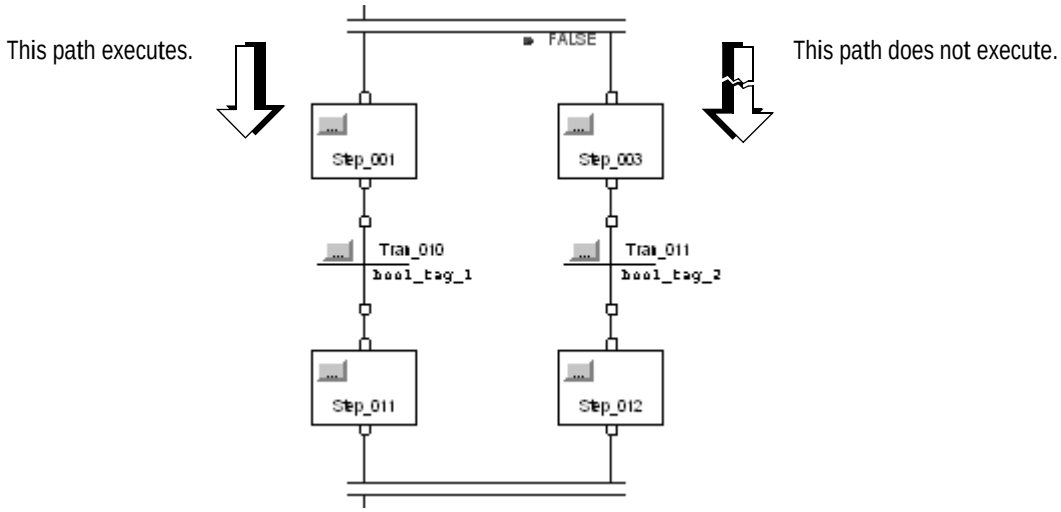


For example, to exit this branch, the SFC must be able to:

- execute Step_011 at least once
- get past Tran_011 and execute Step_012 at least once
- determine that Tran_012 is true

Force a Simultaneous Path

To prevent the execution of a path of a simultaneous branch, force the path false. When the SFC reaches the branch, it executes only the un-forced paths.



If you force a path of a simultaneous branch to be false, the SFC stays in the simultaneous branch as long as the force is active (installed and enabled).

- To leave a simultaneous branch, the last step of each path must execute at least one time and the transition below the branch must be true.
- Forcing a path false prevents the SFC from entering a path and executing its steps.
- When you remove or disable the force, the SFC can execute the steps in the path.

Add an SFC Force

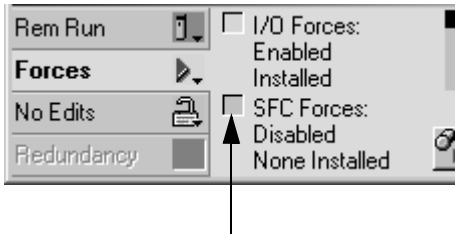
To override the logic of an SFC, use an SFC force:

ATTENTION



Forcing can cause unexpected machine motion that could injure personnel. Before you use a force, determine how the force will effect your machine or process and keep personnel away from the machine area.

- Enabling SFC forces causes your machine or process to go to a different state or phase.
- If forces are enabled and you install a force, the new force immediately takes effect.



1. What is the state of the SFC Forces indicator?

If:	Then note:
off	No SFC forces currently exist.
flashing	No SFC forces are active. But at least one force already exists in your project. When you enable SFC forces, all existing SFC forces will also take effect.
solid	SFC forces are enabled (active). When you install (add) a force, it immediately takes effect.

2. Open the SFC routine.
3. Right-click the transition or start of a simultaneous path that you want to force, and choose either Force TRUE (only for a transition) or Force FALSE.
4. Are SFC forces enabled? (See step 1.)

If:	Then:
no	From the Logic menu, choose SFC Forcing > Enable All SFC Forces. Then choose Yes to confirm.
yes	Stop.

Remove or Disable Forces

ATTENTION



Changes to forces can cause unexpected machine motion that could injure personnel. Before you disable or remove forces, determine how the change will effect your machine or process and keep personnel away from the machine area.

If you want to:	And:	Then:
stop an individual force	leave other forces enabled and in effect	Remove an Individual Force
stop all I/O forces but leave all SFC forces active	leave the I/O forces in the project	Disable All I/O Forces
	remove the I/O forces from the project	Remove All I/O Forces
stop all SFC forces but leave all I/O forces active	leave the SFC forces in the project	Disable All SFC Forces
	remove the SFC forces from the project	Remove All SFC Forces

Remove an Individual Force

ATTENTION



If you remove an individual force, forces remain in the enabled state and any new force immediately takes effect.

Before you remove a force, determine how the change will effect your machine or process and keep personnel away from the machine area.

1. Open the routine that contains the force that you want to remove.
2. What is the language of the routine?

If:	Then:
SFC	Go to step 4.
ladder logic	Go to step 4.
Function Block	Go to step 3.
Structured Text	Go to step 3.

3. Right-click the tag that has the force and choose Monitor…
   If necessary, expand the tag to show the value that is forced (e.g., BOOL value of a DINT tag).
4. Right-click the tag or element that has the force and choose Remove Force.

Disable All I/O Forces

From the Logic menu, choose I/O Forcing > Disable All I/O Forces.
Then choose Yes to confirm.

Remove All I/O Forces

From the Logic menu, choose I/O Forcing > Remove All I/O Forces.
Then choose Yes to confirm.

Disable All SFC Forces

From the Logic menu, choose SFC Forcing > Disable All SFC Forces. Then choose Yes to confirm.

Remove All SFC Forces

From the Logic menu, choose SFC Forcing > Remove All SFC Forces. Then choose Yes to confirm.

Handle a Major Fault

Using This Chapter

Use this chapter to develop logic that handles specific fault conditions.

Topic	Page
Develop a Fault Routine	399
Programmatically Clear a Major Fault	403
Clear a Major Fault During Prescan	405
Test a Fault Routine	408
Create a User-Defined Major Fault	409
Major Fault Codes	412

Develop a Fault Routine

If a fault condition occurs that is severe enough for the controller to shut down, the controller generates a **major fault** and stops the execution of logic.

- Depending on your application, you may not want all major faults to shut down your entire system.
- In those situations, you can use a fault routine to clear a specific fault and let at least some of your system continue to operate.

EXAMPLE Use a fault routine

In a system that uses recipe numbers as indirect addresses, a miss-typed number could produce a major fault, such as type 4, code 20.

To keep the entire system from shutting down, a fault routine clears any type 4, code 20, major faults.

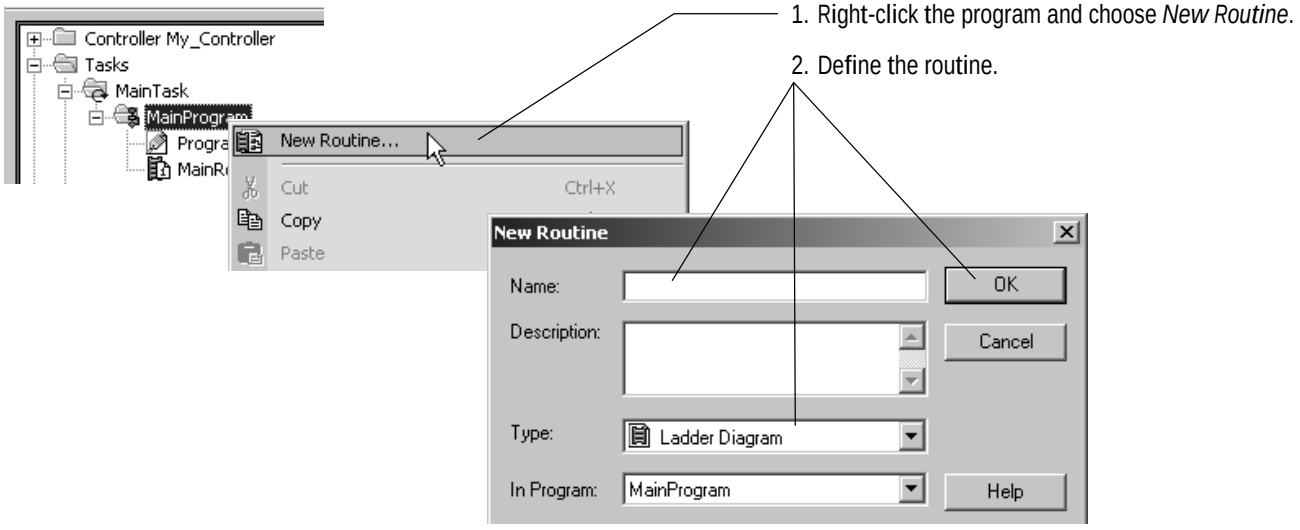
Choose Where to Place the Fault Routine

A fault routine lets you program logic to take specific action after a fault, such as clear the fault and resume execution. Where you place the routine depends on the type of fault that you want to handle:

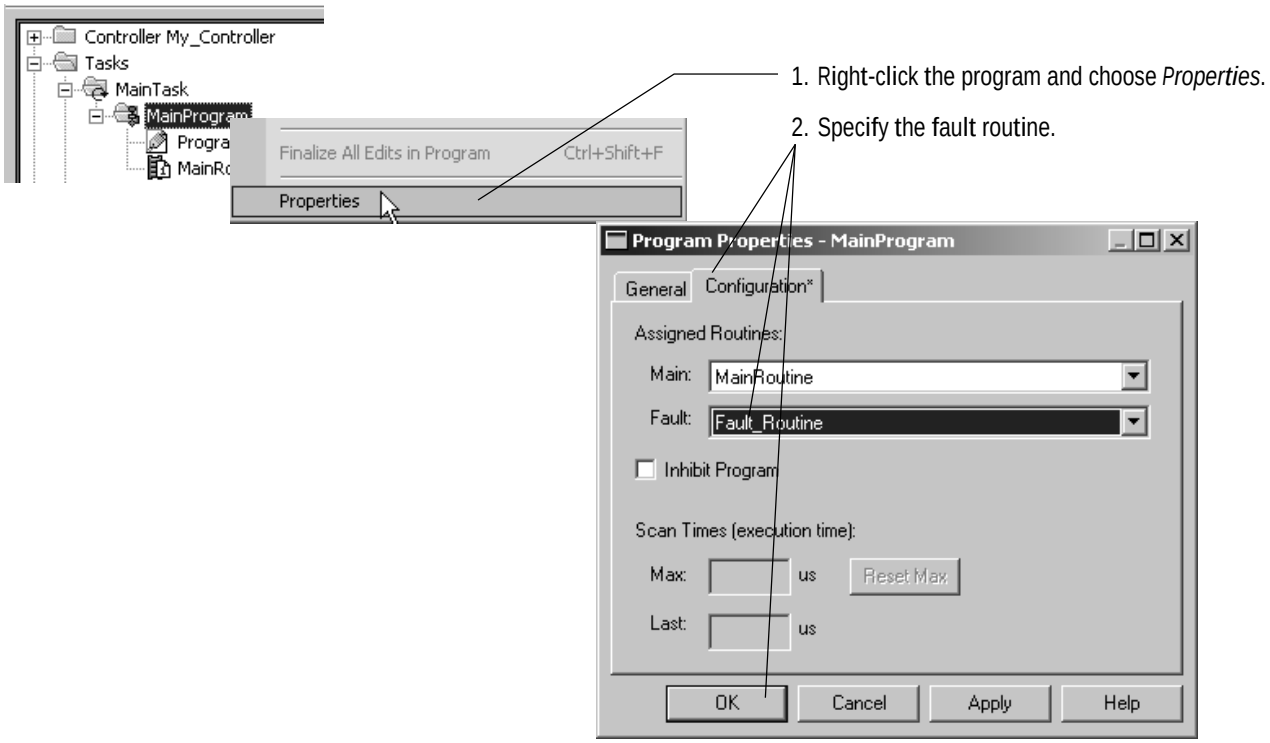
If you want take specific action/clear the fault when:		Do this:	Page
Condition:	**Fault type:**		
The execution of an instruction faults	4	Create a Fault Routine for a Program	400
Communication with an I/O module fails	3	Create a Routine for the Controller Fault Handler	401
Watchdog time for a task expires	6		
While a project is downloading to the controller, the keyswitch is placed in RUN	8		
A motion axis faults	11		
The controller powers up in run/remote run mode	1	Create a Routine for the Power-Up Handler	402

Create a Fault Routine for a Program

Create the Routine



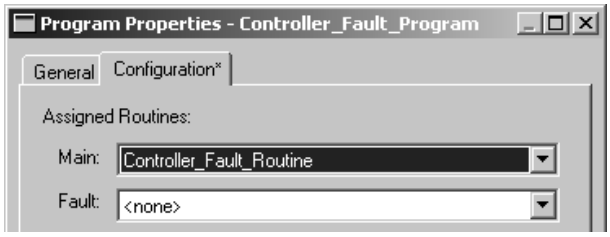
Assign the Routine as the Fault Routine



Create a Routine for the Controller Fault Handler

1. Create a program for the Controller Fault Handler.
2. Create a routine for the program.





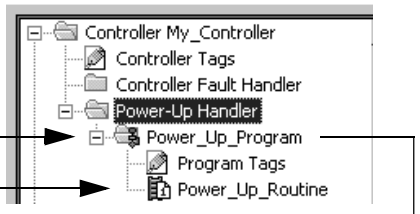
3. Configure the routine as the main routine for the program.

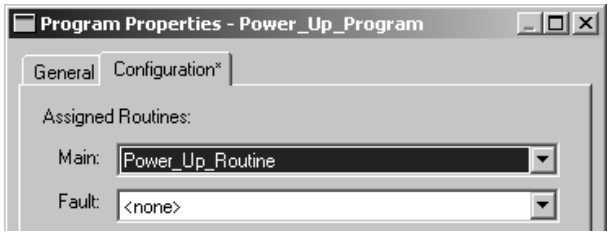
Create a Routine for the Power-Up Handler

The **power-up handler** is an optional task that executes when the controller powers up in run/remote run mode. Use the power-up handler when you want to accomplish either of the following after power is lost and then restored:

To:	Do this:
Prevent the controller from returning to run/remote mode	Leave the routine for the Power-Up Handler empty. When power is restored, a major fault (type 1, code 1) occurs and the controller enters the Faulted mode.
When power is restored, take specific actions and then resume normal operation	In the routine for the Power-Up Handler: 1. Clear the major fault (type 1, code 1). 2. Enter the logic for the actions.

1. Create a program for the Power-Up Handler.
2. Create a routine for the program.





3. Configure the routine as the main routine for the program.

Programmatically Clear a Major Fault

To clear a major fault that occurs during the execution of your project, complete these actions in the appropriate routine. (See Choose Where to Place the Fault Routine on page 16-400.)

- Create a Data Type to Store Fault Information
- Get the Fault Type and Code
- Check for a Specific Fault
- Clear the Fault

Create a Data Type to Store Fault Information

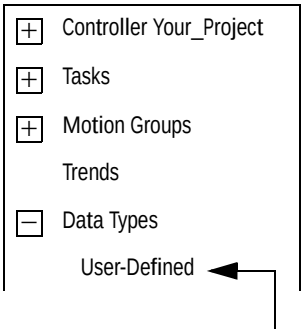
Logix5000 controllers store system information in objects. Unlike PLC-5 or SLC 500 controllers, there is no status file.

- To access system information, you use a Get System Value (GSV) or Set System Value (SSV) instruction.
- For status information about a program, you access the PROGRAM object.
- For fault information, you access these attribute of the PROGRAM object.

Attribute:	Data Type:	Instruction:	Description:
MajorFaultRecord	DINT[11]	GSV SSV	Records major faults for this program Specify the program name to determine which PROGRAM object you want. (Or specify THIS to access the PROGRAM object for the program that contains the GSV or SSV instruction.)

To simplify access to the MajorFaultRecord attribute, create this user-defined data type:

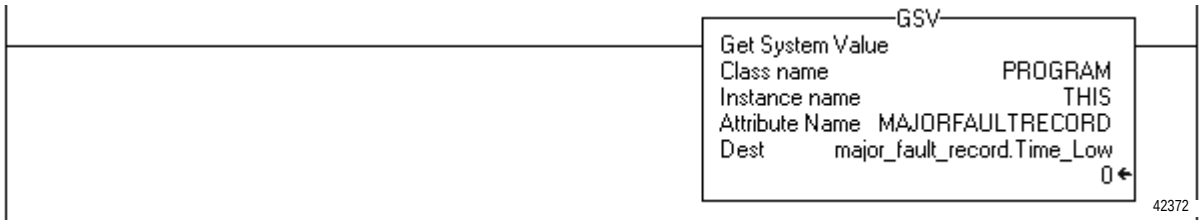
To create a new data type:



Right-click and choose *New Data Type*.

Data Type: FAULTRECORD				
Name	FAULTRECORD			
Description	Stores the MajorFaultRecord attribute or MinorFaultRecord attribute of the PROGRAM object.			
Members				
	Name	**Data Type**	**Style**	**Description**
	Time_Low	DINT	Decimal	lower 32 bits of the fault timestamp value
	Time_High	DINT	Decimal	upper 32 bits of the fault timestamp value
	Type	INT	Decimal	fault type (program, I/O, etc)
	Code	INT	Decimal	unique code for the fault
	Info	DINT[8]	Hex	fault specific information

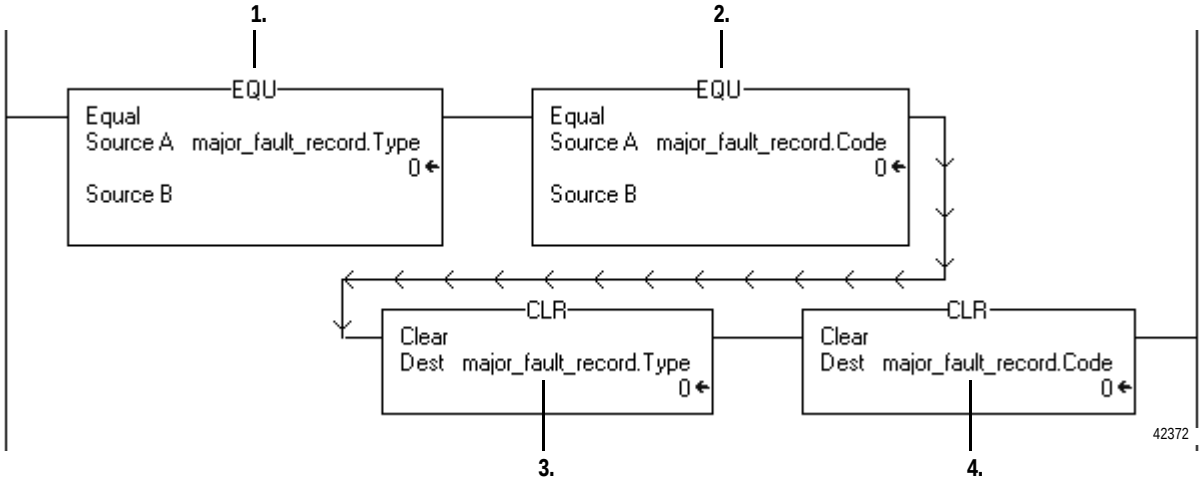
Get the Fault Type and Code



1. The GSV instruction accesses the MAJORFAULTRECORD attribute of this program. This attribute stores information about the fault.

2. The GSV instruction stores the fault information in the major_fault_record tag (of type FAULTRECORD). When you enter a tag that is based on a structure, enter the first member of the tag.

Check for a Specific Fault



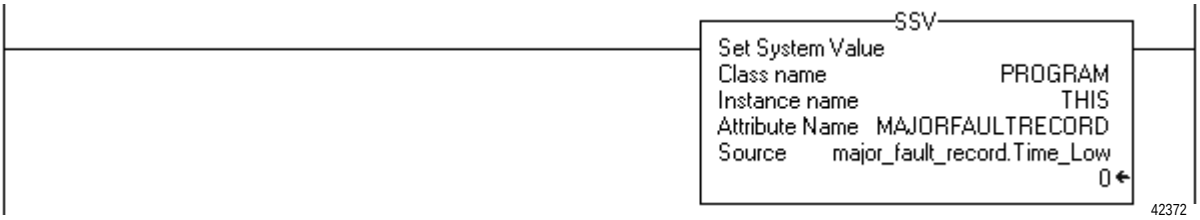
1. The first EQU instruction checks for a specific type of fault, such as program, I/O. In Source B, enter the value for the type of fault that you want to clear.

2. The second EQU instruction checks for a specific fault code. In Source B, enter the value for the code that you want to clear.

3. The first CLR instruction sets to zero the value of the fault type in the major_fault_record tag.

4. The second CLR instruction sets to zero the value of the fault code in the major_fault_record tag.

Clear the Fault



1. The SSV instruction writes new values to the MAJORFAULTRECORD attribute of this program.

2. The SSV instruction writes the values contained in the major_fault_record tag. Since the Type and Code member are set to zero, the fault clears and the controller resumes execution.

Clear a Major Fault During Prescan

If the controller faults immediately after you switch it to the Run mode, then examine the **prescan** operation for the fault. Depending on the revision of your controller, an array subscript that is beyond the range of the array (out of range) during prescan may or may not produce a fault:

If your controller is revision:	Then:
11.x or earlier	During prescan, an array subscript that is beyond the range of the array (out of range) produces a major fault.
12.x	See the release notes for the firmware of your controller.
13.0 or later	During prescan, the controller automatically clears any faults due to an array subscript that is beyond the range of the array (out of range).

To clear a major fault that occurs during prescan:

- Identify When the Controller is in Prescan
- Get the Fault Type and Code
- Check for a Specific Fault
- Clear the Fault

IMPORTANT It is a good programming practice to check for a specific fault before clearing the fault.

Identify When the Controller is in Prescan

In the main routine of your program, enter this rung as the first rung in the main routine of the program:



The fault routine of this program uses the status of this bit to determine if the fault occurred during prescan or normal scan of the logic:

- During prescan, this bit is off. (During prescan, the controller resets all bits that are referenced by OTE instructions.)
- Once the controller begins to execute the logic, the CPU_scanning bit is always on.

Get the Fault Type and Code

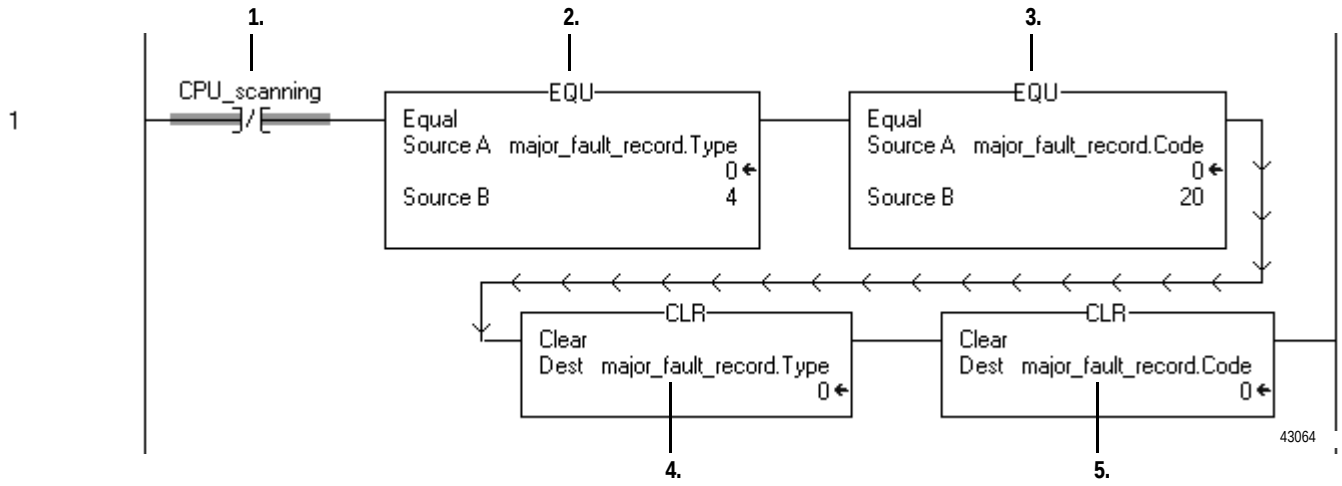
Enter this rung in the fault routine for the program:



1. The GSV instruction accesses the MAJORFAULTRECORD attribute of this program. This attribute stores information about the fault.
2. The GSV instruction stores the fault information in the major_fault_record (of type FAULTRECORD) tag. When you enter a tag that is based on a structure, enter the first member of the tag.

Check for a Specific Fault

Enter this rung in the fault routine for the program:



1. During prescan the bits of all OTE instructions are off and this instruction is true. Once the controller begins to execute the logic, this instruction is always false.

2. The first EQU instruction checks for a fault of type 4, which means that an instruction in this program caused the fault.

3. The second EQU instruction checks for a fault of code 20, which means that either an array subscript is too large, or a POS or LEN value of a CONTROL structure is invalid.

4. The first CLR instruction sets to zero the value of the fault type in the major_fault_record tag.

5. The second CLR instruction sets to zero the value of the fault code in the major_fault_record tag.

Clear the Fault

Enter this rung in the fault routine for the program:



1. During prescan the bits of all OTE instructions are off and this instruction is true. Once the controller begins to execute the logic, this instruction is always false.

2. The SSV instruction writes new values to the MAJORFAULTRECORD attribute of this program.

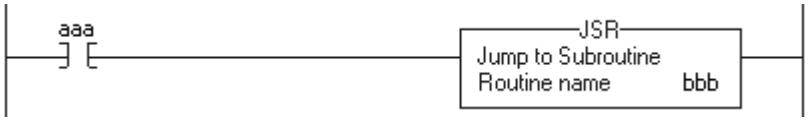
3. The SSV instruction writes the values contained in the major_fault_record tag. Since the Type and Code member are set to zero, the fault clears and the controller resumes execution.

Test a Fault Routine

You can use a JSR instruction to test the fault routine of a program without creating an error (i.e., simulate a fault):

1. Create a BOOL tag that you will use to initiate the fault.

2. In the main routine or a subroutine of the program, enter this rung:

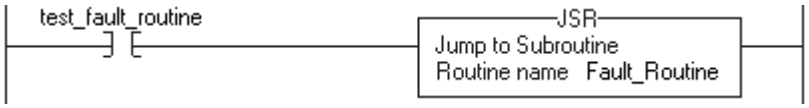


where:	is the:
aaa	tag that you will use to initiate the fault
bbb	fault routine of the program

3. To simulate a fault, set the input condition.

EXAMPLE Test a fault routine

When test_fault_routine is on, a major fault occurs and the controller executes Fault_Routine.



Create a User-Defined Major Fault

If you want to suspend (shut down) the controller based on conditions in your application, create a user-defined major fault. With a user-defined major fault:

- The fault type = 4.
- You define a value for the fault code. Choose a value between 990 to 999. These codes are reserved for user-defined faults.
- The controller handles the fault the same as other major faults:
  - The controller changes to the **faulted mode** (major fault) and stops executing the logic.
  - Outputs are set to their configured state or value for faulted mode.

EXAMPLE User-defined major fault

When Tag_1.0 = 1, produce a major fault and generate a fault code of 999.

To create a user-defined major fault:

- Create a Fault Routine for the Program
- Configure the Program to Use the Fault Routine
- Jump to the Fault Routine

Create a Fault Routine for the Program

Does a fault routine already exist for the program?

If:	Then:
Yes	Go to "Jump to the Fault Routine"on page 16-410
No	Create a fault routine for the program:

1. In the controller organizer, right-click the program and choose New Routine.
2. In the name box, type a name for the fault routine.
3. From the Type drop-down list, choose Ladder.
4. Click OK.

Configure the Program to Use the Fault Routine

1. In the controller organizer, right-click the program and choose New Routine.
2. Select the Configuration tab.
3. From the Fault drop-down list, choose the fault routine.
4. Click OK.

Jump to the Fault Routine

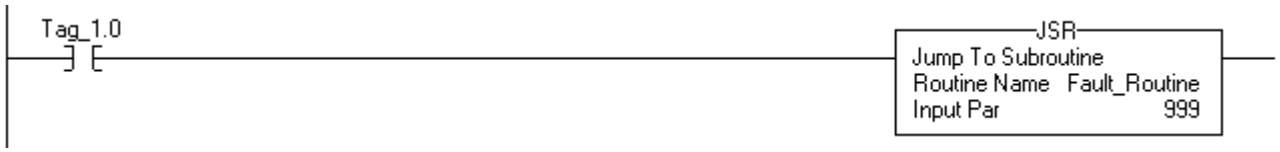
In the main routine of the program, enter this rung:



where:	is:
`Fault_Routine`	name of the fault routine for the program
`999`	value for the fault code

EXAMPLE Create a User-Defined Major Fault

When Tag_1.0 = 1, execution jumps to name_of_fault_routine. A major fault occurs and the controller enters the faulted mode. Outputs go to the faulted state. The Controller Properties dialog box, Major Faults tab, displays the code 999.



Major Fault Codes

The type and code correspond to the type and code displayed in these locations:

- Controller Properties dialog box, Major Faults tab
- PROGRAM object, MAJORFAULTRECORD attribute

Table 16.1 Major Fault Types and Codes

Type:	Code:	Cause:	Recovery Method:
1	1	The controller powered on in Run mode.	Execute the power-loss handler.
1	60	For a controller with *no* CompactFlash card installed, the controller: • detected a non-recoverable fault • cleared the project from memory	1. Clear the fault. 2. Download the project. 3. Change to remote run/run mode. If the problem persists: 1. Before you cycle power to the controller, record the state of the OK and RS232 LEDs. 2. Contact Rockwell Automation support. See the back of this publication.
1	61	For a controller with a CompactFlash card installed, the controller: • detected a non-recoverable fault • wrote diagnostic information to the CompactFlash card • cleared the project from memory	1. Clear the fault. 2. Download the project. 3. Change to remote run/run mode. If the problem persists, contact Rockwell Automation support. See the back of this publication.
3	16	A required I/O module connection failed.	Check that the I/O module is in the chassis. Check electronic keying requirements. View the controller properties Major Fault tab and the module properties Connection tab for more information about the fault.
3	20	Possible problem with the ControlBus chassis.	Not recoverable - replace the chassis.
3	23	At least one required connection was not established before going to Run mode.	Wait for the controller I/O light to turn green before changing to Run mode.
4	16	Unknown instruction encountered.	Remove the unknown instruction. This probably happened due to a program conversion process.
4	20	Array subscript too big, control structure .POS or .LEN is invalid.	Adjust the value to be within the valid range. Don't exceed the array size or go beyond dimensions defined.
4	21	Control structure .LEN or .POS < 0.	Adjust the value so it is > 0.
4	31	The Parameters of the JSR instruction do not match those of the associated SBR or RET instruction.	Pass the appropriate number of Parameters. If too many Parameters are passed, the extra ones are ignored without any error.
4	34	A timer instruction has a negative preset or accumulated value.	Fix the program to not load a negative value into timer preset or accumulated value.
4	42	JMP to a label that did not exist or was deleted.	Correct the JMP target or add the missing label.
4	82	A sequential function chart (SFC) called a subroutine and the subroutine tried to jump back to the calling SFC. Occurs when the SFC uses either a JSR or FOR instruction to call the subroutine.	Remove the jump back to the calling SFC.

Table 16.1 Major Fault Types and Codes (Continued)

Type:	Code:	Cause:	Recovery Method:
4	83	The data tested was not inside the required limits.	Modify value to be within limits.
4	84	Stack overflow.	Reduce the subroutine nesting levels or the number of Parameters passed.
4	89	In a SFR instruction, the target routine does not contain the target step.	Correct the SFR target or add the missing step.
6	1	Task watchdog expired. User task has not completed in specified period of time. A program error caused an infinite loop, or the program is too complex to execute as quickly as specified, or a higher priority task is keeping this task from finishing.	Increase the task watchdog, shorten the execution time, make the priority of this task "higher," simplify higher priority tasks, or move some code to another controller.
7	40	Store to nonvolatile memory failed.	1. Try again to store the project to nonvolatile memory. 2. If the project fails to store to nonvolatile memory, replace the memory board.
7	41	Load from nonvolatile memory failed due to controller type mismatch.	Change to a controller of the correct type or download the project and store it on the CompactFlash card.
7	42	Load from nonvolatile memory failed because the firmware revision of the project in nonvolatile memory does not match the firmware revision of the controller.	Update the controller firmware to the same revision level as the project that is in nonvolatile memory.
7	43	Load from nonvolatile memory failed due to bad checksum.	Contact Rockwell Automation support. See the back of this publication.
7	44	Failed to restore processor memory.	Contact Rockwell Automation support. See the back of this publication.
8	1	Attempted to place controller in Run mode with keyswitch during download.	Wait for the download to complete and clear fault.
11	1	Actual position has exceeded positive overtravel limit.	Move axis in negative direction until position is within overtravel limit and then execute Motion Axis Fault Reset.
11	2	Actual position has exceeded negative overtravel limit.	Move axis in positive direction until position is within overtravel limit and then execute Motion Axis Fault Reset.
11	3	Actual position has exceeded position error tolerance.	Move the position within tolerance and then execute Motion Axis Fault Reset.
11	4	Encoder channel A, B, or Z connection is broken.	Reconnect the encoder channel then execute Motion Axis Fault Reset.
11	5	Encoder noise event detected or the encoder signals are not in quadrature.	Fix encoder cabling then execute Motion Axis Fault Reset.
11	6	Drive Fault input was activated.	Clear Drive Fault then execute Motion Axis Fault Reset.
11	7	Synchronous connection incurred a failure.	First execute Motion Axis Fault Reset. If that doesn't work, pull servo module out and plug back in. If all else fails replace servo module.

Table 16.1 Major Fault Types and Codes (Continued)

Type:	Code:	Cause:	Recovery Method:
11	8	Servo module has detected a serious hardware fault.	Replace the module.
11	9	Asynchronous Connection has incurred a failure.	First execute Motion Axis Fault Reset. If that doesn't work, pull servo module out and plug back in. If all else fails replace servo module.
11	32	The motion task has experienced an overlap.	The group's course update rate is too high to maintain correct operation. Clear the group fault tag, raise the group's update rate, and then clear the major fault.

Monitor Minor Faults

When to Use This Chapter

If a fault condition occurs that is not severe enough for the controller to shut down, the controller generates a **minor fault**.

- The controller continues to execute.
- You do not need to clear a minor fault.
- To optimize execution time and ensure program accuracy, you should monitor and correct minor faults.

Monitor Minor Faults

To use Ladder logic to capture information about a minor fault:

To check for a	Do this
periodic task overlap	1. Enter a GSV instructions that gets the FAULTLOG object, MinorFaultBits attribute. 2. Monitor bit 6.
load from nonvolatile memory	1. Enter a GSV instructions that gets the FAULTLOG object, MinorFaultBits attribute. 2. Monitor bit 7.
problem with the serial port	1. Enter a GSV instructions that gets the FAULTLOG object, MinorFaultBits attribute. 2. Monitor bit 9.
low battery	1. Enter a GSV instructions that gets the FAULTLOG object, MinorFaultBits attribute. 2. Monitor bit 10.
problem with an instruction	1. Create a user-defined data type that stores the fault information. Name the data type FaultRecord and assign these members:

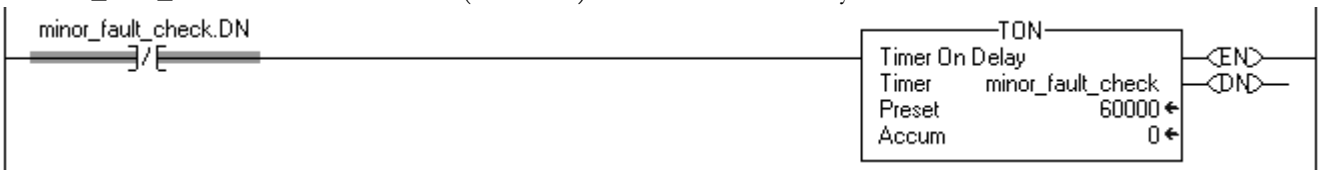
Name:	Data Type:	Style:
TimeLow	DINT	Decimal
TimeHigh	DINT	Decimal
Type	INT	Decimal
Code	INT	Decimal
Info	DINT[8]	Hex

2. Create a tag that will store the values of the MinorFaultRecord attribute. Select the data type from step 1.
3. Monitor S:MINOR.
4. If S:MINOR is on, use a GSV instruction to get the values of the MinorFaultRecord attribute.
5. If you want to detect a minor fault that is caused by another instruction, reset S:MINOR. (S:MINOR remains set until the end of the scan.)

This example checks for a low battery warning.

EXAMPLE Check for a minor fault

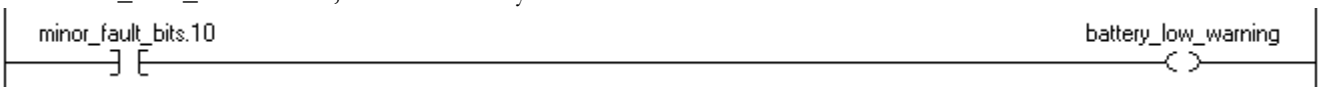
Minor_fault_check times for 1 minute (60000 ms) and then automatically restarts itself.



Every minute, minor_fault_check.DN turns on for one scan. When this occurs, the GSV instruction gets the value of the FAULTLOG object, MinorFaultBits attribute, and stores it in the minor_fault_bits tag. Because the GSV instruction only executes once every minute, the scan time of most scans is reduced.



If minor_fault_bits.10 is on, then the battery is low.



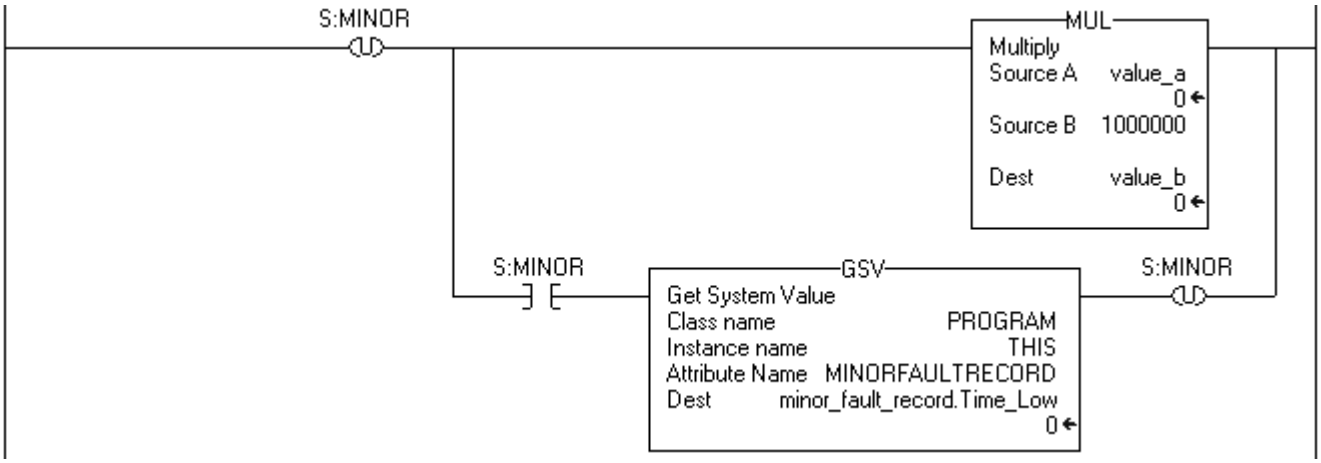
42373

This example checks for a minor fault that is caused by a specific instruction.

EXAMPLE Check for a minor fault that is caused by an instruction

Multiplies value_a by 1000000 and checks for a minor fault, such as a math overflow:

- To make sure that a previous instruction did not produce the fault, the rung first clears S:MINOR.
- The rung then executes the multiply instruction.
- If the instruction produces a minor fault, the controller sets S:MINOR.
- If S:MINOR is set, the GSV instruction gets information about the fault and resets S:MINOR.



Minor Fault Codes

The type and code correspond to the type and code displayed in these locations:

- Controller Properties dialog box, Minor Faults tab
- PROGRAM object, MINORFAULTRECORD attribute

Table 17.1 Minor Fault Types and Codes

Type:	Code:	Cause:	Recovery Method:
4	4	An arithmetic overflow occurred in an instruction.	Fix program by examining arithmetic operations (order) or adjusting values.
4	5	In a GSV/SSV instruction, the specified instance was not found.	Check the instance name.
4	6	In a GSV/SSV instruction, either: • specified Class name is *not* supported • specified Attribute name is *not* valid	Check the Class name and Attribute name.
4	7	The GSV/SSV destination tag was too small to hold all of the data.	Fix the destination so it has enough space.
4	35	PID delta time ≤ 0.	Adjust the PID delta time so that it is > 0.
4	36	PID setpoint out of range	Adjust the setpoint so that it is within range.
4	51	The LEN value of the string tag is greater than the DATA size of the string tag.	1. Check that no instruction is writing to the LEN member of the string tag. 2. In the LEN value, enter the number of characters that the string contains.
4	52	The output string is larger than the destination.	Create a new string data type that is large enough for the output string. Use the new string data type as the data type for the destination.
4	53	The output number is beyond the limits of the destination data type.	Either: • Reduce the size of the ASCII value. • Use a larger data type for the destination.
4	56	The Start or Quantity value is invalid.	1. Check that the Start value is between 1 and the DATA size of the Source. 2. Check that the Start value plus the Quantity value is less than or equal to the DATA size of the Source.
4	57	The AHL instruction failed to execute because the serial port is set to no handshaking.	Either: • Change the Control Line setting of the serial port. • Delete the AHL instruction.
6	2	Periodic task overlap. Periodic task has not completed before it is time to execute again.	Simplify program(s), or lengthen period, or raise relative priority, etc.
7	49	Project loaded from nonvolatile memory.	
9	0	Unknown error while servicing the serial port.	Contact GTS personnel.

Table 17.1 Minor Fault Types and Codes (Continued)

Type:	Code:	Cause:	Recovery Method:
9	1	The CTS line is not correct for the current configuration.	Disconnect and reconnect the serial port cable to the controller. Make sure the cable is wired correctly
9	2	Poll list error. A problem was detected with the DF1 master's poll list, such as specifying more stations than the size of the file, specifying more then 255 stations, trying to index past the end of the list, or polling the broadcast address (STN #255).	Check for the following errors in the poll list: • total number of stations is greater than the space in the poll list tag • total number of stations is greater than 255 • current station pointer is greater than the end of the poll list tag • a station number greater than 254 was encountered
9	5	DF1 slave poll timeout. The poll watchdog has timed out for slave. The master has not polled this controller in the specified amount of time.	Determine and correct delay for polling.
9	9	Modem contact was lost. DCD and/or DSR control lines are not being received in proper sequence and/or state.	Correct modem connection to the controller.
10	10	Battery not detected or needs to be replaced.	Install new battery.

Notes:

Access Status Information

When to Use This Chapter

Logix5000 controllers do not have a status file, as in the PLC-5 controller. To access status information, you use a keyword or access a specific object.

To:	Page
Monitor Status Flags	421
Get and Set System Data	422

Monitor Status Flags

The controller supports status keywords you can use in your logic to monitor specific events:

- The status keywords are not case sensitive.
- Because the status flags can change so quickly, RSLogix 5000 software does not display the status of the flags. For example, even when a status flag is set, an instruction that references that flag is not highlighted.
- You cannot define a tag alias to a keyword.

You can use these key words:

To determine if:	Use:
the value you are storing cannot fit into the destination because it is either: • greater than the maximum value for the destination • less than the minimum value for the destination **Important:** Each time S:V goes from cleared to set, it generates a minor fault (type 4, code 4)	S: V
the instruction's destination value is 0	S:Z
the instruction's destination value is negative	S:N
an arithmetic operation causes a carry or borrow that tries to use bits that are outside of the data type For example: • adding 3 + 9 causes a carry of 1 • subtracting 25 - 18 causes a borrow of 10	S:C
this is the first, normal scan of the routines in the current program	S:FS
at least one minor fault has been generated: • The controller sets this bit when a minor fault occurs due to program execution. • The controller does not set this bit for minor faults that are not related to program execution, such as battery low.	S:MINOR

The Status of S:FS When the Project Has an SFC

The state of S:FS depends on the status of the SFC:

- If you use S:FS in an action of a sequential function chart (SFC), S:FS is set (on) for one scan each time the step goes active. S:FS = *step_name*.FS.

EXAMPLE SFC calls a Ladder diagram

Suppose several steps in an SFC call the same Ladder diagram routine. And suppose the Ladder diagram uses S:FS. Each time one of those steps goes active, S:FS turns on for one scan of the Ladder diagram.

- If the SFC calls a routine, S:FS is set (on) for one scan each time the step that calls the routine goes active. S:FS = *step_name*.FS.

  If the SFC does not call a routine, S:FS is set (on) for the first scan of the task.

EXAMPLE Several tasks but no SFC

Suppose you have 2 tasks that use Ladder diagrams. When the first task runs for the first time, S:FS turns on for one scan. After that, S:FS stays off for that task. When the other task runs for the first time, S:FS turns on for one scan in that task. S:FS stays off in the first task that ran.

For more information on S:FS, see page 129.

Get and Set System Data

The controller stores system data in objects. There is no status file, as in the PLC-5 controller. Use the GSV/SSV instructions get and set controller system data that is stored in objects:

- The GSV instruction retrieves the specified information and places it in the destination.
- The SSV instruction sets the specified attribute with data from the source.

ATTENTION Use the SSV instruction carefully. Making changes to objects can cause unexpected controller operation or injury to personnel.



To get or set a system value:

1. Open the RSLogix 5000 project.
2. From the Help menu, select Contents.
3. Select the Index tab.
4. Type gsv/ssv objects and click Display.
5. Select the object.

To get or set:	Click:
axis of a servo module	AXIS
system overhead timeslice	CONTROLLER
physical hardware of a controller	CONTROLLERDEVICE
coordinated system time for the devices in one chassis	CST
DF1 communication driver for the serial port	DF1
fault history for a controller	FAULTLOG
attributes of a message instruction	MESSAGE
status, faults, and mode of a module	MODULE
group of axes	MOTIONGROUP
fault information or scan time for a program	PROGRAM
instance number of a routine	ROUTINE
configuration of the serial port	SERIALPORT
properties or elapsed time of a task	TASK
wall clock time of a controller	WALLCLOCKTIME

6. In the list of attributes for the object, identify the attribute that you want to access.

7. Create a tag for the value of the attribute:

If the data type of the attribute is:	Then:
one element (e.g., DINT)	Create a tag for the attribute.
more than one element (e.g., DINT[7])	A. Create a user-defined data type that matches the organization of data that is used by the attribute. B. Create a tag for the attribute and use the data type from Step A..

8. In the Ladder Logic routine, enter the appropriate instruction:

To:	Enter this instruction:
get the value of an attribute	GSV
set the value of an attribute	SSV

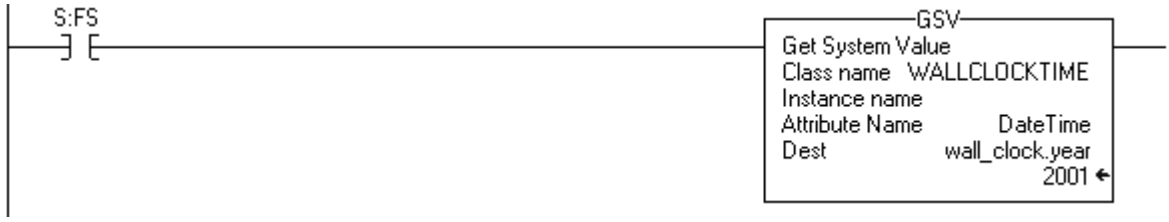
9. Assign the required operands to the instruction:

For this operand:	Select:
Class name	name of the object
Instance name	name of the specific object (e.g., name of the required I/O module, task, message) • Not all objects require this entry. • To specify the current task, program, or routine, select THIS.
Attribute Name	name of the attribute
Dest (GSV)	tag that will store the retrieved value • If the tag is a user-defined data type or an array, select the first member or element.
Source (SSV)	tag that stores the value to be set • If the tag is a user-defined data type or an array, select the first member or element.

This examples gets the current date and time.

EXAMPLE Get a system value

At the first scan, gets the DateTime attribute of the WALLCLOCKTIME object and stores it in the wall_clock tag, which is based on a user-defined data type.



42370

For more information, see the *Logix5000 Controllers General Instruction Set Reference Manual*, publication 1756-RM003.

Security and RSLogix 5000 Software

Introduction

This chapter discusses the various ways you can secure a project. Refer to the the RSLogix 5000 software and FactoryTalk online help for more detailed information.

Topic	Page
Understanding the Current Relationship Between FactoryTalk Security and RSI Security Server	425
About Migrating RSI Security Server to FactoryTalk Security	426
Differences between RSI Security Server and FactoryTalk Security	427
Setting Up Source Protection	428
Using Routine Source Protection	431
Using FactoryTalk Security to Protect a Project	442
Migrating to FactoryTalk Security	442
Using the Security Server	443

Understanding the Current Relationship Between FactoryTalk Security and RSI Security Server

In earlier versions of RSLogix 5000 software security actions were grouped into four built-in action groups. The groups were called:

- View Project
- Go Online
- Maintain Project
- Full Access

These groups could be found under the RSLogix 5000 Actions/Groups folder in the RSI Security server. You could view these action groups. They represented sets of individual actions within RSLogix 5000 software.

In RSLogix 5000 software Revision 15, many of the internal actions were made available to you so they could be individually set as well as being able to use the action groups.

About Migrating RSI Security Server to FactoryTalk Security

If you currently use RSI Security Server, consider migrating to FactoryTalk Security.

- FactoryTalk Security offers centralized administration.

  Instead of maintaining lists of user, computer, and group accounts in each product, and then duplicating these lists in RSSecurity Server, FactoryTalk Security lets you to define users, computers, and groups only once, and then maintain them in one place.

- A central authority for security.

  Instead of defining portions of your security system in individual products, and then portions of the system in RSI Security Services, FactoryTalk Security lets you to define security for all FactoryTalk products in one place. Users can log on and be authenticated by one central authority -- the FactoryTalk Directory -- regardless of which product they use to access the FactoryTalk system.

- Get your system running, or make changes, quickly without duplication of effort.

  RSSecurity Server requires setting up individual products, and then defining securable actions and resources for them, such as reading or writing values in data servers. With FactoryTalk Security, a complete list of securable actions is available automatically for every product that supports FactoryTalk Security.

How FactoryTalk Security Differs from RSI Security Server

Depending on which features of RSI Security Server you are using, it might not be possible to migrate everything in your RSI Security Server database to FactoryTalk Security. This is because of differences in the way that RSI Security Server and FactoryTalk work. Refer to the FactoryTalk Automation Services 2.00 online help for more information.

Differences between RSI Security Server and FactoryTalk Security

This section discusses the some of the differences between RSI Security Server and FactoryTalk Security.

Support for resource groups and action groups

With FactoryTalk Automation Platform 2.00, action groups and resource grouping is not supported. The process imports the RSSecurity actions and resources held within groups as individual items. If you have created resource and action groups, they are not created groups when you migrate to FactoryTalk Automation Platform 2.00.

IMPORTANT If you rely on these groups, we recommend waiting to migrate until FactoryTalk Services Platform 2.10 becomes available.

In FactoryTalk Services Platform 2.10, as long as all of your computers in a FactoryTalk system have been upgraded to FactoryTalk Services Platform 2.10 or later, you can retain RSI Security Services group memberships. The import process prompts you to choose an application or area as a destination for the resources held in a group. Imported RSI Security Server action groups appear under Action Groups in the FactoryTalk Directory System folder.

Unsupported features

Some RSI Security Server features are not supported by FactoryTalk Security. Refer to the FactoryTalk Automation Services 2.00 online help for detailed information.

Setting Up Source Protection

Follow these steps to set up source protection for component. The process is the same whether you are protecting a routine or an Add-On Instruction.

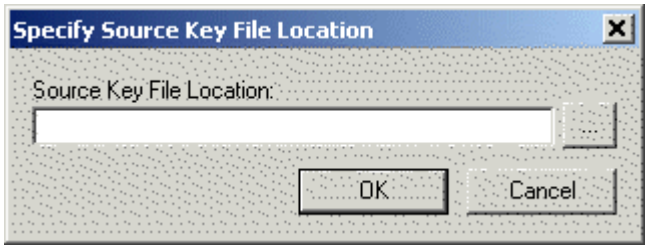
1. In RSLogix 5000 select Tools -> Security -> Configure Source Protection, to access the source configuration dialog.

TIP

You may find that Source Protection is unavailable, not listed in your menus.

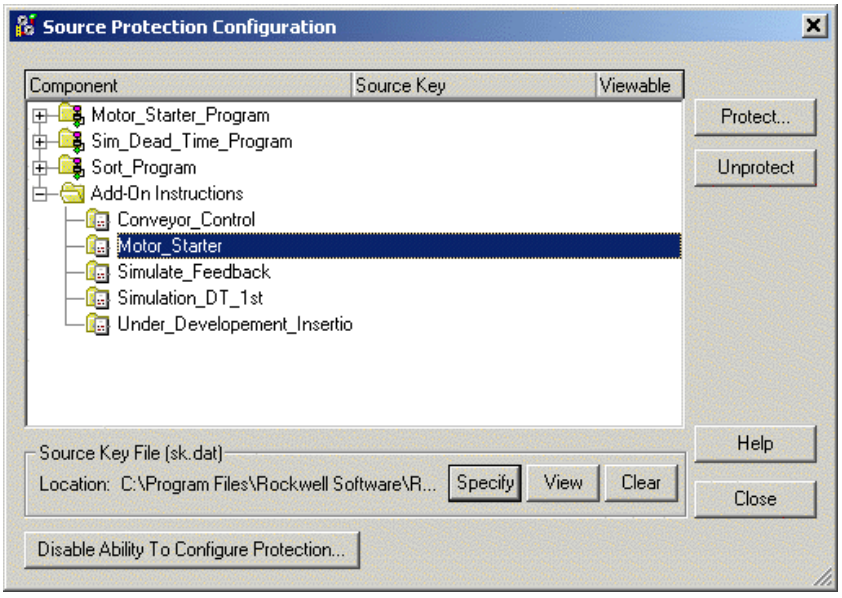
To enable the Source Protection feature, enable it during the RSLogix 5000 System Install or use the tool RS5KSrcPtc.exe on your installation CD.

If it is the first time the Source Protection has been configured, a dialog appears asking to configure a source key file location.



2. Click Browse to specify the location for the Source Key File.
3. Click OK in both dialogs to accept.
4. Click Yes to confirm creation of the sk.dat Key File.

The Source Protection Configuration dialog appears.



Source Protect Components

Follow these steps to source protect a component The components include all Routines and Add-On Instruction definitions in the project.

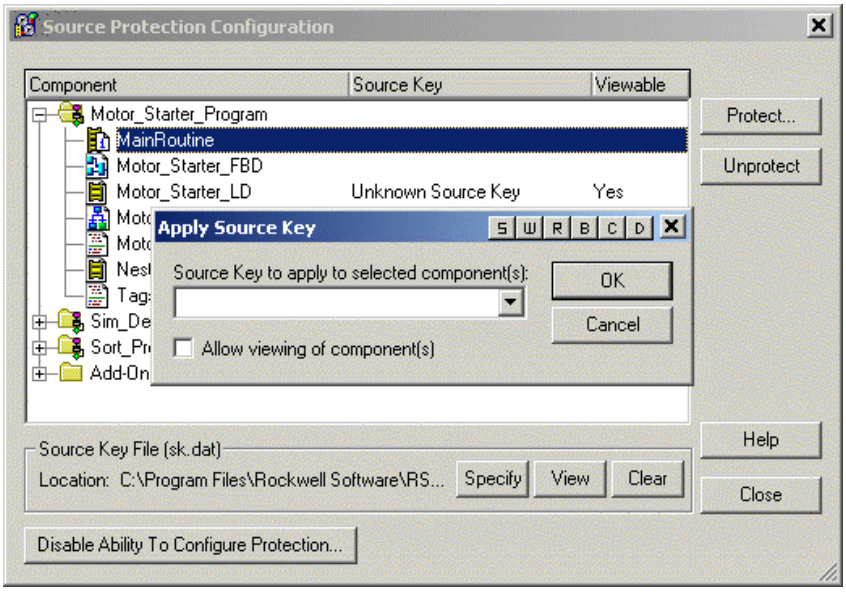
1. Expand the folder to view the component the you want to source protect.

2. Select the component to protect and click Protect.

   The Source Key Entry dialog appears. This is where you enter the individual source key for this component. These source keys follow the conventions for routine source keys.

   Refer to Define the Source Key or Keys on page 435 for more information.

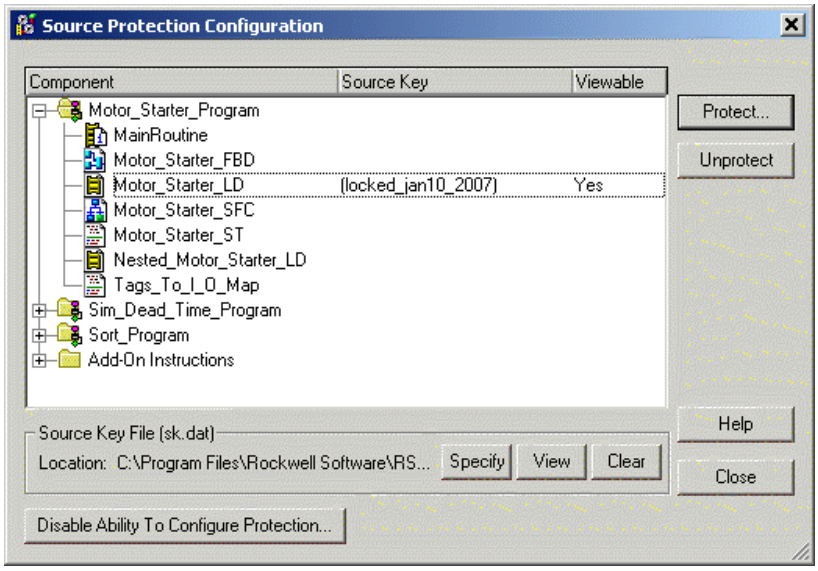
3. Enter a name for the source key for this component or use an existing one.



4. If you wish to allow users to be able to see the logic, check the box Allow viewing of component(s).

5. Click OK.

The item is now protected with the file you selected.



IMPORTANT Before you click Close, you must first either click Clear or Disable Ability to Configure Source Protection or remove the sk.dat file from the PC so that the source key is no longer present, if you wish to observe the source protection settings.

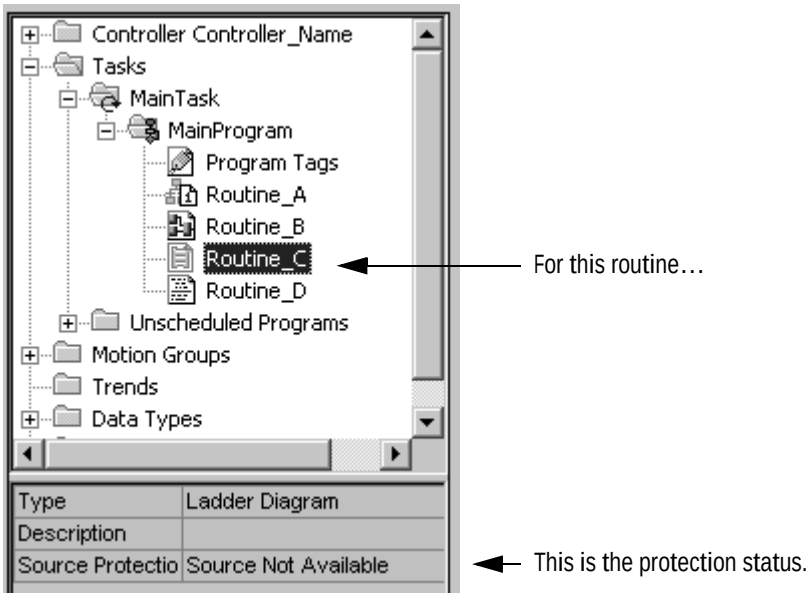
6. Click Close.
7. Save the project.
8. Close RSLogix 5000 software.

Using Routine Source Protection

To limit who has access to a routine, use RSLogix 5000 software to assign a security key to the routine (protect the routine).

- To protect a routine, you have to first activate the feature for RSLogix 5000 software.
- Once you protect a routine, a computer requires the source key to edit, copy, or export the routine.
- You have the option of making a routine either viewable or not viewable without the source key.
- Regardless of whether or not the source key is available, you can always download the project and execute all the routines.
- You can gain access by either of these methods:
  - manually enter the name(s) of the source keys into a source file that you create or that already exists.
  - point RSLogix 5000 software to the location of the source key file.

The controller organizer shows the protection status of a routine:



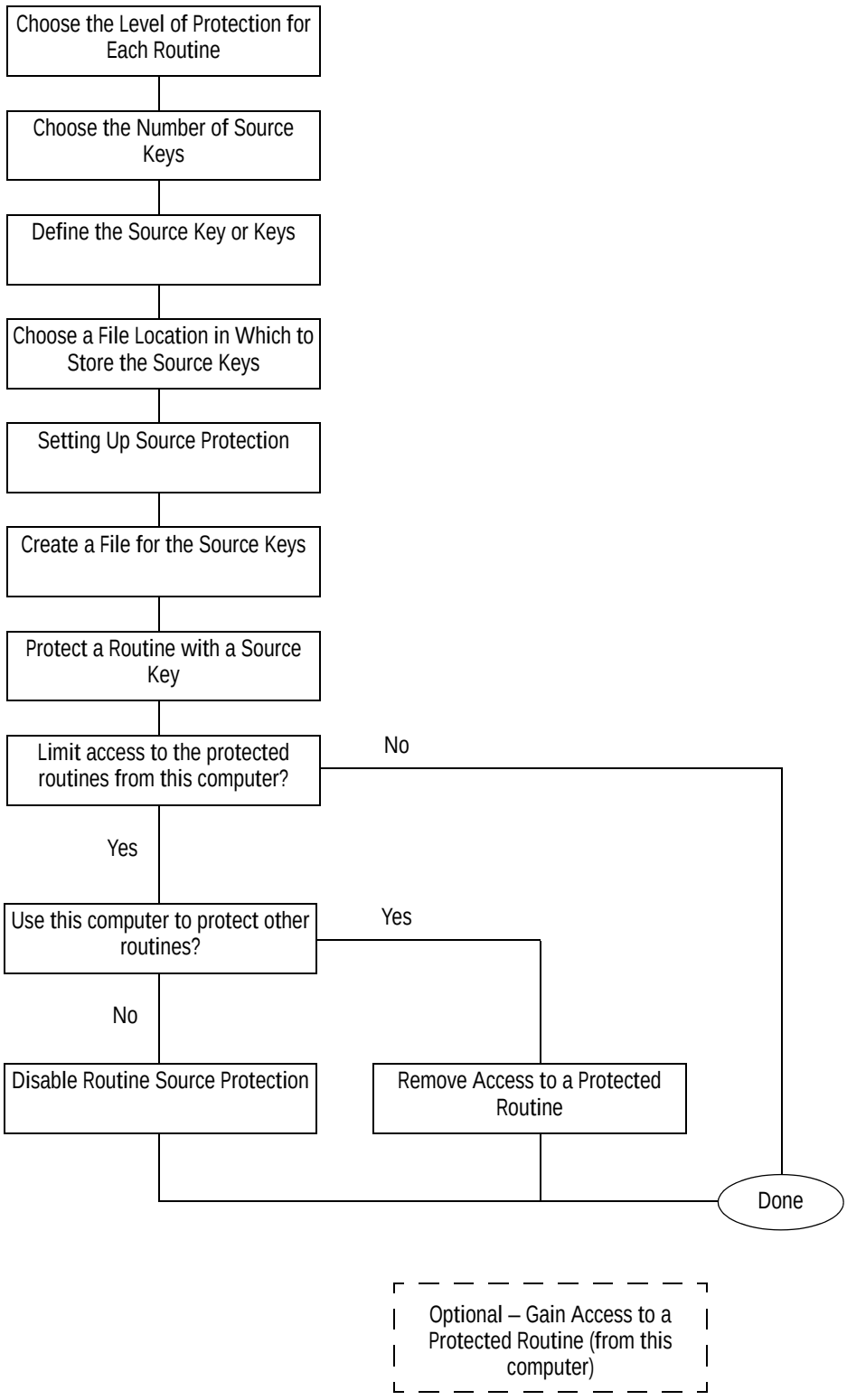
If the controller organizer displays	Then
Source Not Available	• A source key is assigned to the routine. • To open the routine, your computer requires the source key for the routine.
Source Not Available (Viewable)	• A source key is assigned to the routine. • You can only open and view the routine. • You cannot make any changes or copy any of contents of the routine.

If the controller organizer displays	Then
Source Available	• A source key is assigned to the routine. • You have full access to the routine.
Source Available (Viewable)	• A source key is assigned to the routine. • You have full access to the routine. • Those who do not have the source key can still view the routine.
none of the above	• No source key is assigned to the routine. • You have full access to the routine.

IMPORTANT If the source of a routine is unavailable, do not export the project.

- An export file (.L5K) contains only routines where the source code is available.
- If you export a project where the source code is not available for all routines, you will not be able to restore the entire project.

To assign and manage source keys, you:



Choose the Level of Protection for Each Routine

Source protection protects your project at the routine level.

Routine Protection Options

If You Want To	And	Then	
		Protect the routine?	Allow viewing?
Prevent someone from doing this: • edit the routine • change the properties of the routine • export the routine	also prevent someone from doing this: • open (display) the routine • search the routine • go to cross references within the routine • print the routine	yes	no
	no other limitations	yes	yes
Let anyone have full access to the routine	→	no	

Choose the Number of Source Keys

To protect a routine, you assign a source key to the routine. You can reuse a source key as often as you like, as shown below.

This	Gives You
one source key for all projects	fewest number of source keys (easier to manage but less protection)
unique source key for each project	
unique source key for each routine in each project	greatest number of source keys (more difficult to manage but more protection)

Choose the number of source keys that balances your need for protection verses the level of source key management that you want to undertake.

Define the Source Key or Keys

Source keys follow the same rules for names as other RSLogix 5000 software components, such as routines, tags, and modules. A source key:

- must begin with an alphabetic character (A-Z or a-z) or an underscore (_)
- can contain only alphabetic characters, numeric characters, and underscores
- can have as many as 40 characters
- must not have consecutive or trailing underscore characters (_)
- are not case sensitive

Choose a File Location in Which to Store the Source Keys

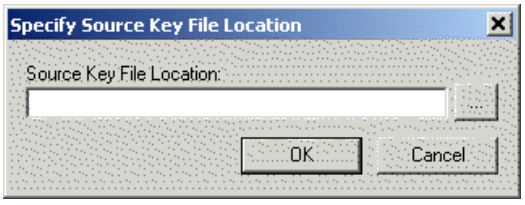
A source key file (sk.dat) stores the source keys. The source key file is separate from the RSLogix 5000 project files (.acd). You can store the source key file in any folder that you choose.

Create a File for the Source Keys

1. Open the RSLogix 5000 project that you want to protect.
2. From the Tools menu, choose Security > Configure Source Protection.
3. Does RSLogix 5000 software prompt you to specify the location for the source key file?

If	Then
No	Your computer already has the source key file. Go to “Protect a Routine with a Source Key“on 436.
Yes	Go to step 4.

4. Choose Yes.

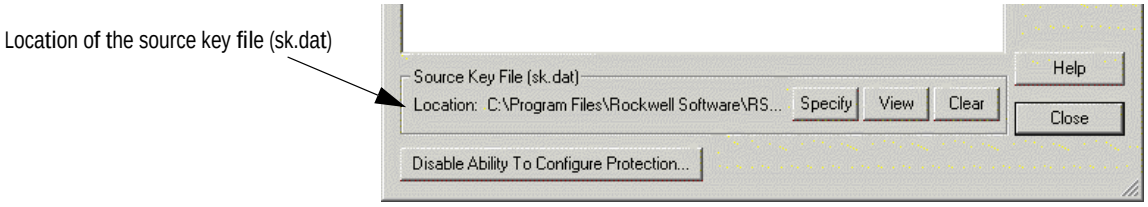


5. Browse to the folder where you want to store the file.

6. Click OK.

   A dialog box asks if you want to create the source key file (sk.dat).

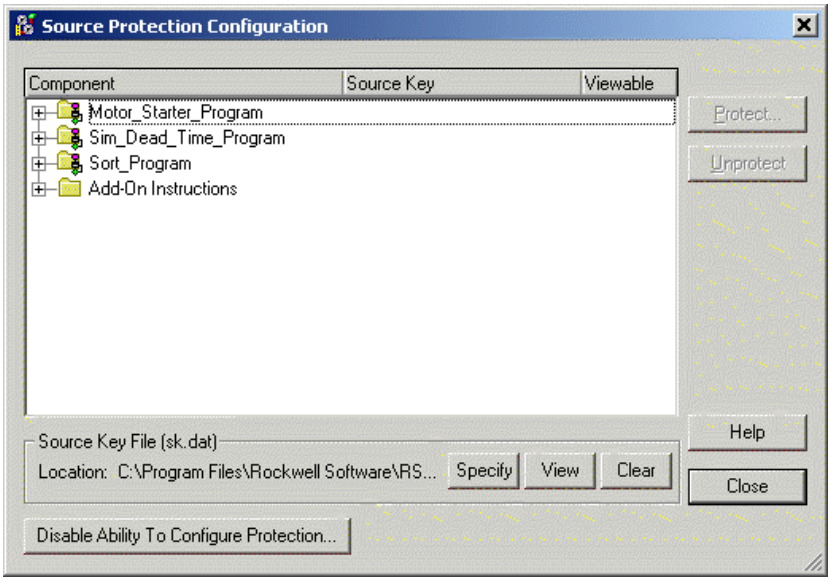
7. Choose Yes.



Protect a Routine with a Source Key

1. Open the RSLogix 5000 project that you want to protect.

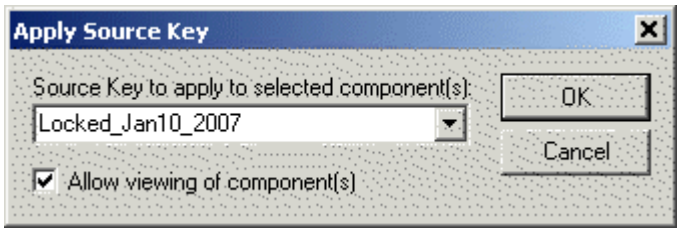
2. From the Tools menu, choose Security >Configure Source Protection.



3. Select the routine or routines that you want to protect.

4. Click Protect.

- Type a name that you want to use as the source key. Or select an existing source key from the drop-down list.



- If someone does not have the source key, do you want to let them open and view the routine?

If	Then
No	Clear (uncheck) the Allow viewing of routine check box (default).
Yes	Check the Allow viewing of routine check box.

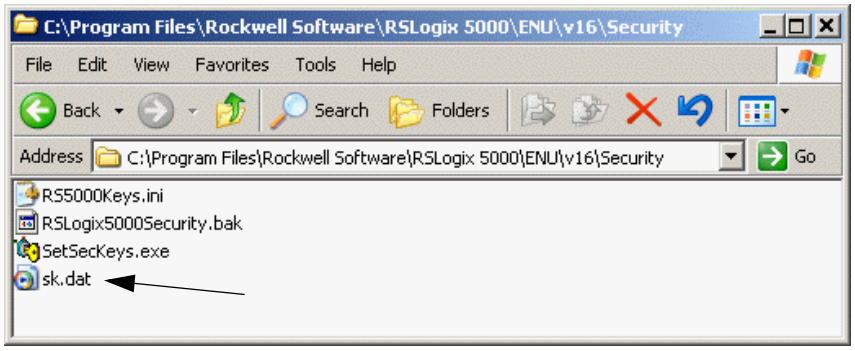
5. Click OK.

6. When you have assigned the required source keys to the project, click Close.

7. From the File menu, choose Save.

View Protected Routine

To view a protected routine, you must have the Source Key for the routine. All source keys are stored in the Source Key file (sk.dat). RSLogix 5000 software looks for the source key file in the same directory in which the programming software application is installed.

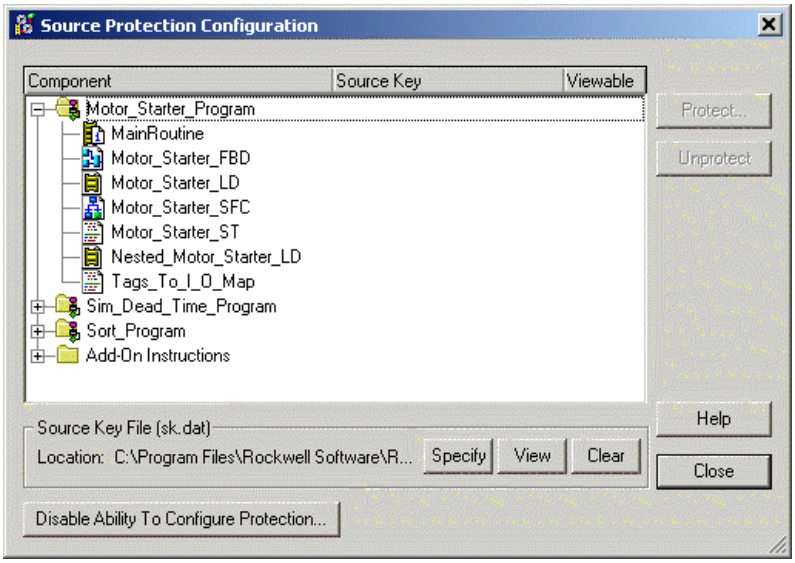


You can also store this file in a different location if you choose.

Remove Access to a Protected Routine

IMPORTANT Before you remove the source key file (sk.dat) from a computer either write down the source keys or make a copy of the file and store it in a secure location.

1. Open the RSLogix 5000 project that is protected.
2. From the Tools menu, choose Security > Configure Source Protection.



3. Click Clear.

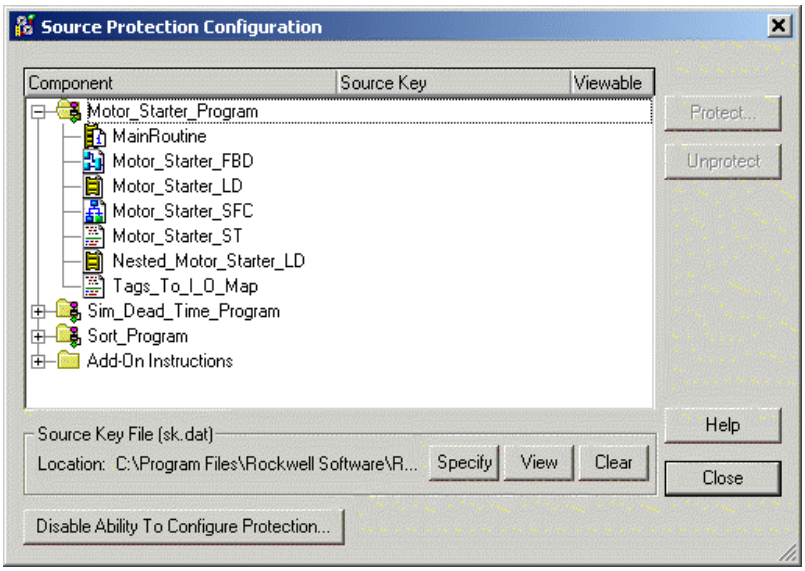
   A dialog box asks if you want to delete the source key file (sk.dat).

4. Select Yes, to remove or No, not to remove the source key file from the computer (prevent future access to the file)?

Disable Routine Source Protection

IMPORTANT Before you remove the source key file (sk.dat) from a computer either write down the source keys or make a copy of the file and store it in a secure location.

1. Open the RSLogix 5000 project that is protected.
2. From the Tools menu, choose Security > Configure Source Protection.



3. Click Disable Ability To Configure Protected Routines.

   A dialog box prompts you to confirm the action.

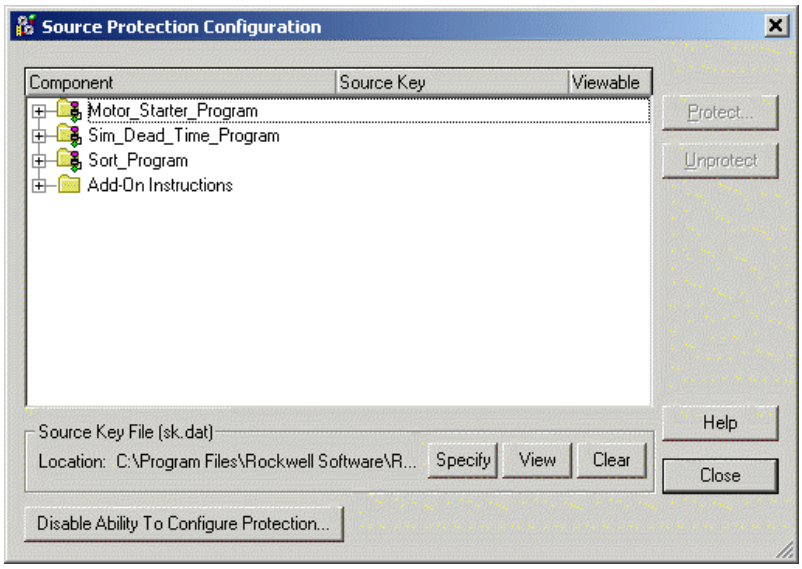
4. Choose Yes.

   A dialog box asks if you want to delete the source key file (sk.dat).

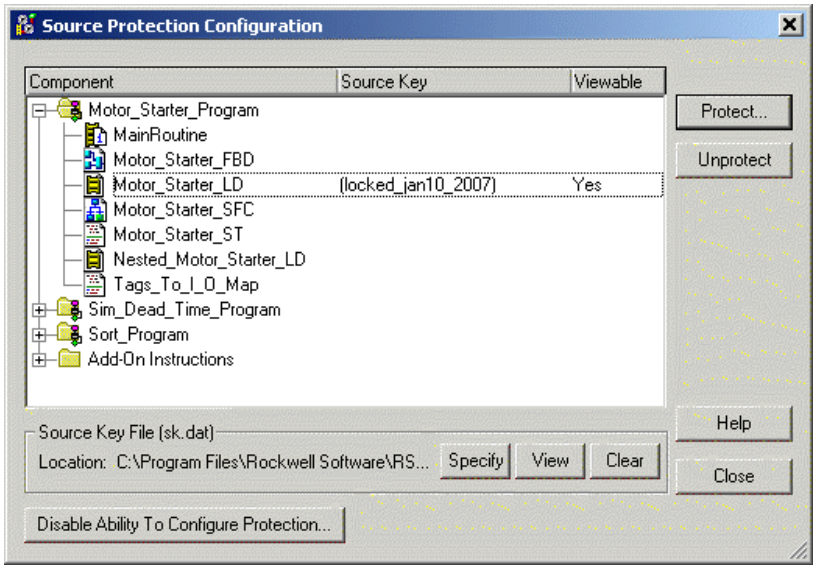
5. Select Yes, to remove or No, not to remove the source key file from the computer (prevent future access to the file)?

Gain Access to a Protected Routine

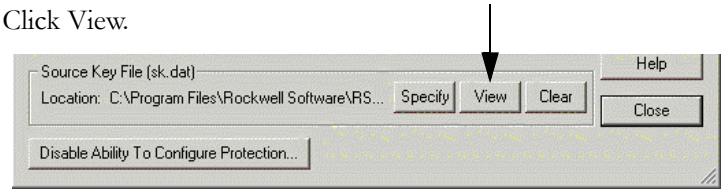
1. Open the RSLogix 5000 project that contains the protected routines.
2. From the Tools menu, choose Security > Configure Source Protection.



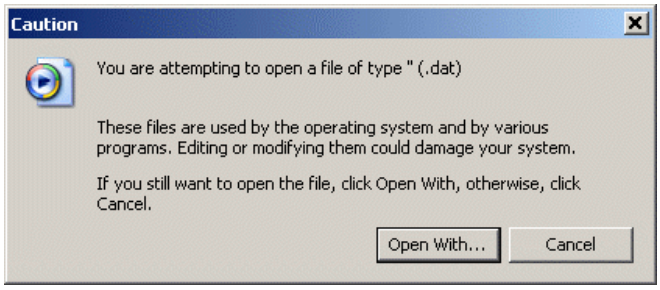
3. Open the routine that contains the program that you want to view.
4. Select the program and click View.



5. Click View.



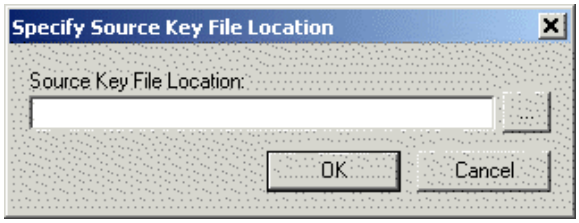
- If you are prompted to select a program with which to open the file, select a word processing program, such as Notepad.



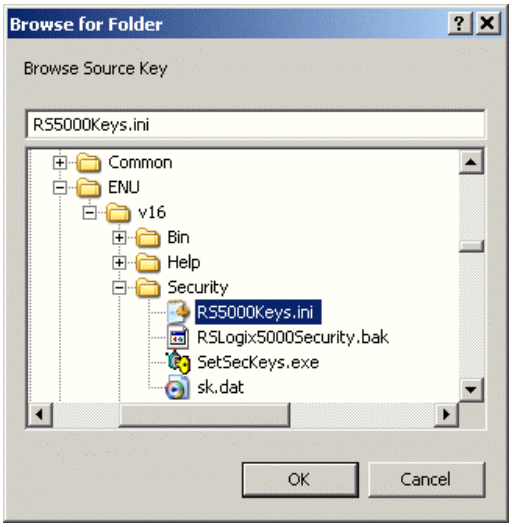
- The sk.dat file opens.

6. Review the file, Save and close the sk.dat file.

7. Choose Yes.



8. Click Browse.



Using FactoryTalk Security to Protect a Project

FactoryTalk Security software lets you control the access that individuals have to RSLogix 5000 projects. Refer to the FactoryTalk online help for more detailed information.

With this software, you customize access to projects based on the:

- user that is currently logged into the workstation.
- RSLogix 5000 project that the user is accessing.
- workstation from which the user is accessing the RSLogix 5000 project.

Migrating to FactoryTalk Security

IMPORTANT If your processes currently rely on resource and action groups we recommend waiting to upgrade to FactoryTalk Service Platform 2.10 when it becomes available. If you upgrade to FTAP 2.00 you will lose all resource and action group configurations and you will have to recreate them in FTSP 2.10.

If you wait to upgrade to FTSP 2.10, your groups will automatically be created when you migrate from RSI Security Server to FTSP 2.10.

1. Back up your existing RSSecurity database. Refer to FTAP 2.00 Online Help for specific instructions.

2. Run FactoryTalk Administration Console and back up your existing FactoryTalk Directory.

IMPORTANT If you are installing V16 and using FTAP 2.00 you need to make sure you are aware of the update utility discussed under the FactoryTalk online help topic Update FactoryTalk Services Platform 2.00 with new RSLogix 5000 actions.

3. Use the FactoryTalk Security Import tool to import an RSSecurity database into the FactoryTalk Directory.

4. Install the RSSecurity Emulator to allow FactoryTalk Directory to emulate an RSSecurity Server.

   This lets your existing RSSecurity clients, for example, the RSLogix 5000 family of products, to connect to FactoryTalk Directory. See Install the RSSecurity Emulator.

5. When the migration is complete, use either FactoryTalk administration Console or FactoryTalk View Studio to view and edit resources.

Using the Security Server

The Security Server supports four built-in action groups specific to RSLogix 5000 software. These action groups combine related actions into groups for easier selection. The action groups are View Project, Go Online, Maintain Project, and Full Access. Granting access to an action group grants access to the associated actions within that group. Each group also builds on the previous group. For example, granting access to Maintain Project also grants access to Go Online and View Project. "Go Online" and "View Project" actions.

Version 16 of RSLogix 5000 software continues the ability to grant access to individual actions with an action group.

These action groups and individual actions apply to RSLogix 5000 software:

To let a user	Grant access to this project (which grants access to all associated actions)	Or grant access to these individual actions associated with the project
Open a (read-only) version of the project. A user must have access to open and view a project in order to do anything else with the project.	View Project	Project: Open
Go online with View Project access	Go Online	Project: Go Online
Access and modify controller information in a project	Maintain Project	Controller: Clear Faults Controller: Modify Mode Controller: Modify Revision Controller: Modify Type
Access and modify module information in a project	Maintain Project	Module: Modify Properties
Print reports	Maintain Project	Print: Report
Access and modify projects	Maintain Project	Project: Compact Project: Download Project: Export Project: Go Online Project: Modify Path Project: Modify Path Project: Save Project: Save As Project: Upload
Access and modify tags	Maintain Project	Tag: Force Tag: Modify Values
Access and modify trends	Maintain Project	Trend: Create Trend: Delete Trend: Modify Properties Trend: Run
Update controller firmware	Maintain Project	Update Firmware
Provide full access to the controller	Full Access	Controller: Lock/Unlock Controller: Modify Properties

To let a user	Grant access to this project (which grants access to all associated actions)	Or grant access to these individual actions associated with the project
Provide full access to modules in the controller organizer	Full Access	Module: Create Module: Delete Module: Maintenance High Module: Maintenance Low
Provide full access to the motion configuration and commands	Full Access	Motion: Command Axis Motion: Modify Configuration
Provide full access to the nonvolatile memory in the controller	Full Access	Nonvolatile Memory: Load Nonvolatile Memory: Store
Provide full access to the equipment phases	Full Access	Phase: Create Phase: Delete Phase: Manual Control Phase: Modify Properties
Provide full access map PLC/SLC tags	Full Access	PLC/SLC: Modify Tag Mappings
Provide full access to programs	Full Access	Program: Create Program: Delete Program: Modify Properties
Provide full access to routines	Full Access	Routine: Create Routine: Delete Routine: Manual Control Routine: Modify Logic Routine: Modify Properties
Provide full access to tags	Full Access	Tag: Create Tag: Delete Tag: Modify Properties
Provide full access to tasks	Full Access	Task: Create Task: Delete Task: Modify Properties
Provide full access to user-defined tags	Full Access	User Defined Type: Create User Defined Type: Delete User Defined Type: Modify
Unsecure a secured controller	N/A	Unsecure Controller

Store and Load a Project Using Nonvolatile Memory

When to Use This Chapter

IMPORTANT Nonvolatile memory stores the contents of the user memory at the time that you store the project.

- Changes that you make after you store the project are not reflected in nonvolatile memory.
- If you make changes to the project but do not store those changes, you overwrite them when you load the project from nonvolatile memory. If this occurs, you have to upload or download the project to go online.
- If you want to store changes such as online edits, tag values, or a ControlNet network schedule, store the project again after you make the changes.

Use this procedure to **store** or **load** a project using the **nonvolatile memory** of a controller.

- If the controller loses power and does not have enough battery capacity, it loses the project in user memory.
- Nonvolatile memory lets you keep a copy of your project on the controller. The controller does not need power to keep this copy.
- You can load the copy from nonvolatile memory to the user memory of the controller:
  - on every power-up
  - whenever there is no project in the controller and it powers-up
  - anytime through RSLogix 5000 software

Topic	Page
Before You Use Nonvolatile Memory	20-446
Store a Project	20-451
Load a Project	20-453
Check for a Load	20-455
Clear Nonvolatile Memory	20-456
Use a CompactFlash Reader	20-458

Before You Use Nonvolatile Memory

A store or load has the following parameters:

Parameter:	Store:	Load:
How much time does a store or load take?	If the controller does *not* use a 1784-CF64 Industrial CompactFlash card, a store may take up to 3 minutes. If the controller uses a CompactFlash card, the store is considerably faster (less than a minute).	several seconds
In what controller mode (s) can I store or load a project?	program mode	
Can I go online with the controller during a store or load?	no	
What is the state of the I/O during a store or load?	I/O remains in its configured state for program mode.	

Choose a Controller That Has Nonvolatile Memory

The following Logix5000 controllers have nonvolatile memory for project storage.

Controller Type:	Catalog #:	Firmware Revision:	Requires a 1784-CF64 Industrial CompactFlash memory card:
CompactLogix	1769-L31	13.x or later	yes
	1769-L32E	13.x or later	yes
	1769-L35CR	13.x or later	yes
	1769-L35E	12.x or later	yes
ControlLogix	1756-L55M22	10.x or later	no
	1756-L55M23	8.x or later	no
	1756-L55M24	8.x or later	no
	1756-L60M03SE	13.x or later	yes
	1756-L61	12.x or later	yes
	1756-L62	12.x or later	yes
	1756-L63	11.x or later	yes
DriveLogix	5720	10.x or later	no
	5730	13.x or later	yes
FlexLogix	1794-L34/B	11.x or later	no

Prevent a Major Fault During a Load

If the major and minor revision of the project in nonvolatile memory does not match the major and minor revision of the controller, a major fault may occur during a load.

If the controller:	Then:
does not use a CompactFlash card	Make sure that the major and minor revision of the project in nonvolatile memory matches the major and minor revision of the controller. The nonvolatile memory of the controller stores only the project. It does not store the firmware for the controller.
uses a CompactFlash card	The CompactFlash card stores the firmware for projects ≥ 12.0. Depending on the current revision of the controller, you may be able to use the CompactFlash card to update the firmware of the controller and load the project. See "Determine How to Handle Firmware Updates" on 448.

Format a CompactFlash Card

When you store a project to a 1784-CF64 Industrial CompactFlash memory card, the controller formats the card, if required.

If the revision of your project is:	Then:	
11.x	The CompactFlash card uses a special format. • Use only a Logix5000 controller to store a project on a CompactFlash card. • Store only a single Logix5000 project and no other data on a CompactFlash card. • When you store a project on a CompactFlash card, you overwrite the entire contents of the card. In other words, you lose everything that is currently on the card.	
≥ 12.0	The CompactFlash card uses the FAT16 file system.	
	If the card:	**Then the controller:**
	is already formatted for the FAT16 file system	• Leaves existing data. • Creates folders and files for the project and firmware.
	is *not* formatted for the FAT16 file system	• Deletes existing data. • Formats the card for the FAT16 file system. • Creates folders and files for the project and firmware.
	Once the CompactFlash card is formatted for the FAT16 file system: • The CompactFlash card stores multiple projects and associated firmware. • If the CompactFlash card already contains a project with same name, a store overwrites the project on the CompactFlash card. • The CompactFlash card loads the most recently stored project. With a revision ≥ 12.0, you can also use a CompactFlash reader to read and manipulate the files on a CompactFlash card. See "Use a CompactFlash Reader" on 458.	

Determine How to Handle Firmware Updates

The following table outlines the options and precautions for updating the firmware of a controller that has nonvolatile memory.

<table>
<tr><th>If:</th><th>Then:</th></tr>
<tr><td>You meet all of the following conditions:
The controller uses a 1784-CF64 Industrial CompactFlash card.
The project on the CompactFlash card has a revision ≥ 12.0.
The project on the CompactFlash card has a Load Image option = On Power Up or On Corrupt Memory.
If the controller is a 1756-L63 controller, its firmware revision is either:
For a controller just out of its box, revision ≥ 1.4. (Look for the F/W REV. on the side of the controller or its box.)
For a controller already in service, revision ≥ 12.0.</td><td>Update the firmware using either:
CompactFlash card
RSLogix 5000 software
ControlFlash software
To update the firmware and load the project using the CompactFlash card:
Install the card in the controller.
If the Load Image option = On Corrupt Memory and the controller contains a project, disconnect the battery from the controller.
Turn on or cycle power to the controller.
If you use RSLogix 5000 software or ControlFlash software to update the firmware:
During the update, the controller sets the Load Image option of the CompactFlash card to User Initiated. To prevent this, remove the card from the controller.
After you update the firmware, store the project again to nonvolatile memory. This ensures that the revision of the project in nonvolatile memory matches the revision of the controller.</td></tr>
<tr><td>You do not meet all of the conditions listed above:</td><td>Update the firmware using either:
RSLogix 5000 software
ControlFlash software
Take these precautions:
Before you update the firmware:<table><tr><th>If the controller:</th><th>Then:</th></tr><tr><td>does not use a CompactFlash card</td><td>Save the project to an offline file. When you update the firmware of the controller, you erase the contents of the nonvolatile memory (revision 10.x or later).</td></tr><tr><td>uses a CompactFlash card</td><td>Either:
Remove the CompactFlash card from the controller.
Check the Load Image option of the CompactFlash card. If it is set to On Power Up or On Corrupt Memory, first store the project with the Load Image option set to User Initiated.
Otherwise, you may get a major fault when you update the firmware of the controller. This occurs because the On Power Up or On Corrupt Memory options cause the controller to load the project from nonvolatile memory. The firmware mismatch after the load then causes a major fault.</td></tr></table>
After you update the firmware, store the project again to nonvolatile memory. This ensures that the revision of the project in nonvolatile memory matches the revision of the controller.</td></tr>
</table>

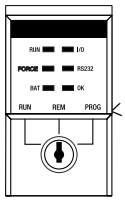
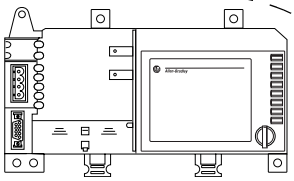
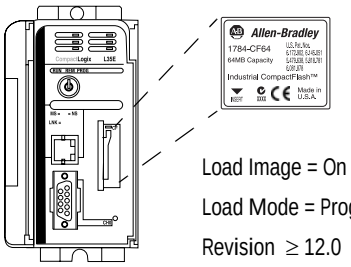
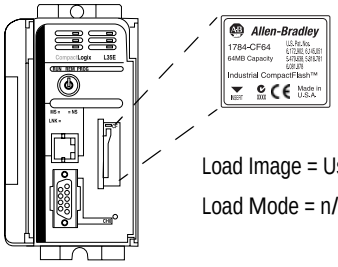
Choose When to Load an Image

You have several options for when (under what conditions) to load the project back into the user memory (RAM) of the controller:

If you want to load it:	Then select:	Notes:
whenever you turn on or cycle the chassis power	On Power Up	• During a power cycle, you will lose any online changes, tag values, and network schedule that you have not stored in the nonvolatile memory. • A 1784-CF64 Industrial CompactFlash card may also change the firmware of the controller. • This occurs if both the revision of the project on the CompactFlash card and the revision of the controller firmware are ≥ 12.0. • For more information, see "Determine How to Handle Firmware Updates" on 448. • You can always use RSLogix 5000 software to load the project.
whenever there is no project in the controller and you turn on or cycle the chassis power	On Corrupt Memory	• For example, if the battery becomes discharged and the controller loses power, the project is cleared from memory. When power is restored, this load option loads the project back into the controller. • A 1784-CF64 Industrial CompactFlash card may also change the firmware of the controller. • This occurs if both the revision of the project on the CompactFlash card and the revision of the controller firmware are ≥ 12.0. • For more information, see "Determine How to Handle Firmware Updates" on 448. • You can always use RSLogix 5000 software to load the project.
only through RSLogix 5000 software	User Initiated	

Examples

Here are some example uses for the different load options:

Example:	Description:
1.  Nonvolatile memory Load Image = On Power Up Load Mode = Program	1. You update the firmware of the controller to the desired revision. 2. You store the project for the controller in nonvolatile memory. 3. When you turn on power to the controller after installation, the project loads into the controller. 4. The controller remains in program mode.
2. Nonvolatile memory Load Image = On Corrupt Memory Load Mode = Run 	1. You store the project for the controller in nonvolatile memory. (The major and minor revision of firmware in the controller match the major and minor revision of the project in nonvolatile memory.) 2. If the battery of the controller becomes completely discharged and power to the controller is interrupted, the project is cleared from controller memory. 3. When power is restored, the project automatically loads into the controller and the controller returns to the run mode.
3.  Load Image = On Power Up Load Mode = Program Revision $\geq$ 12.0	1. The controller fails. 2. You remove the CompactFlash card. 3. You replace the failed controller with a new controller. 4. You replace the CompactFlash card. 5. When you turn on the power, both the firmware and project load into the controller. The controller remains in program mode.
4.  Load Image = User Initiated Load Mode = n/a	1. You want to load a different project into your controller. 2. A CompactFlash card contains the desired project. 3. With the CompactFlash card installed in the controller, you use RSLogix 5000 software to load the project into the controller.

Store a Project

In this task, you store a project in the nonvolatile memory of the controller.

ATTENTION



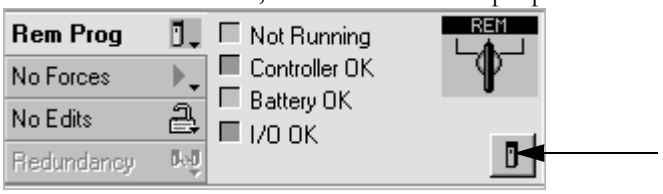
During a store, all active servo axes are turned off. Before you store a project, make sure that this will not cause any unexpected movement of an axis.

Before you store the project:

- make all the required edits to the logic
- download the project to the controller
- schedule your ControlNet networks

To store a project:

1. Go online with the controller.
2. Put the controller in Program mode (Rem Program or Program).
3. On the Online toolbar, click the controller properties button.



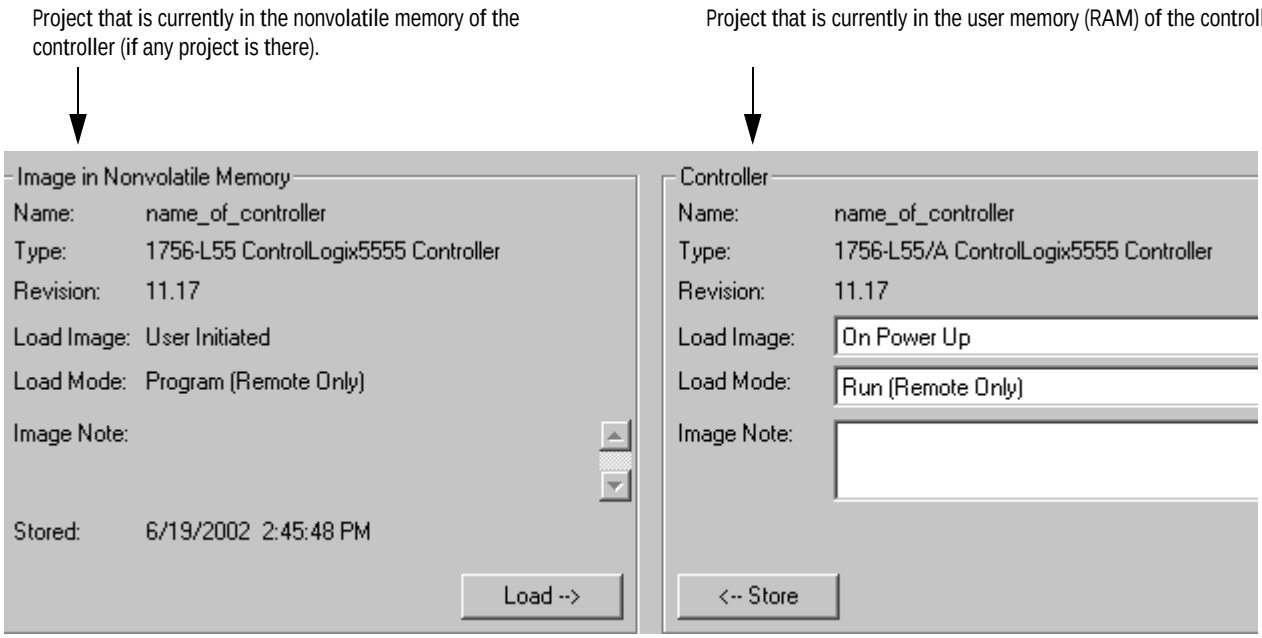
42627

4. Select the Nonvolatile Memory tab and click Load/Store.



42865

5. Select when (under what conditions) to load the project back into the user memory (RAM) of the controller.



If you select On Power Up or On Corrupt Memory, you must also select the mode you want to controller to go to after the load:

- Remote Program
- Remote Run

6. Click <- Store.

A dialog box asks you to confirm the store.

7. To store the project, choose Yes.

During the store, the following events occur:

- On the front of the controller, the OK LED displays the following sequence: flashing green ⇒ solid red ⇒ solid green
- RSLogix 5000 software goes offline.
- A dialog box tells you that the store is in progress.

8. Click OK.

When the store is finished, you remain offline.

Load a Project

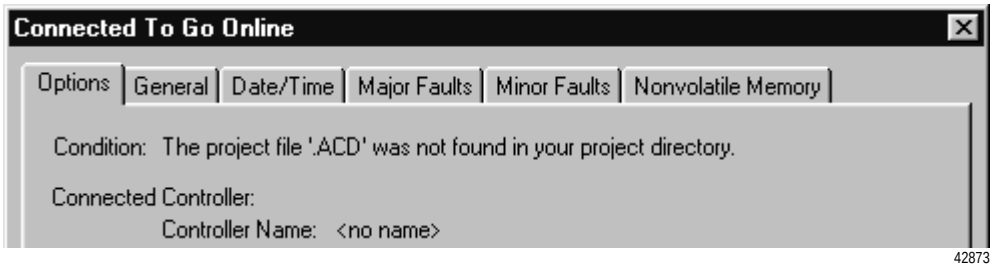
To use RSLogix 5000 software to load the project from nonvolatile memory:.

ATTENTION



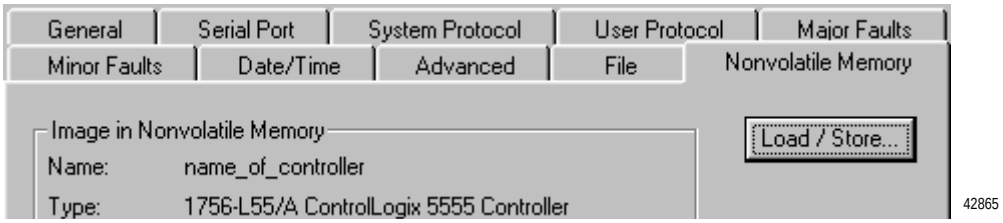
During a load, all active servo axes are turned off. Before you load a project, make sure that this *will not* cause any unexpected movement of an axis.

1. Go online with the controller.
2. Did the following dialog box open?

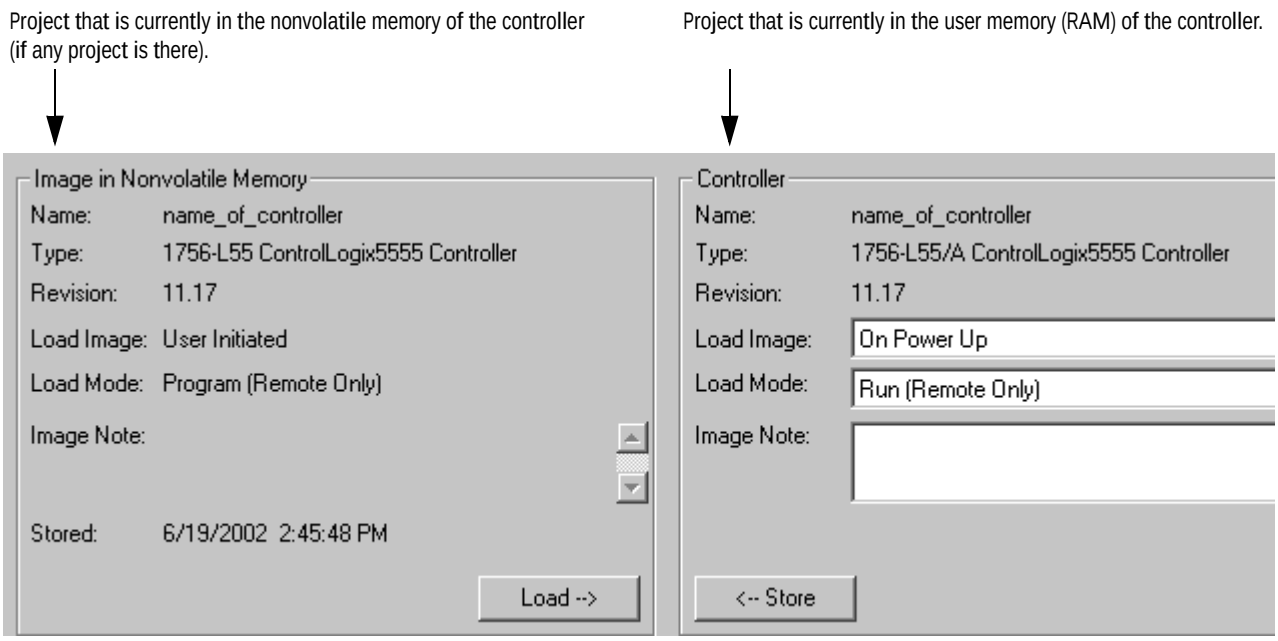


If:	Then:
No	a. Put the controller in Program mode (Rem Program or Program). b. On the Online toolbar, click the controller properties button.
Yes	Put the controller in Program mode (Rem Program or Program). Use either the: • General tab of the Connected To Go Online dialog box. • keyswitch on the front of the controller

3. Select the Nonvolatile Memory tab and click Load/Store.



4. Click Load ->.



A dialog box asks you to confirm the load.

5. To load the project from the nonvolatile memory, click Yes.

During the load, the following events occur:

- On the front of the controller, the OK LED displays this sequence:

If the load:	Then the OK LED displays:
does not include firmware	solid red > solid green
includes firmware	flashing red > solid red > solid green

- RSLogix 5000 software goes offline.

When the load is finished, you remain offline.

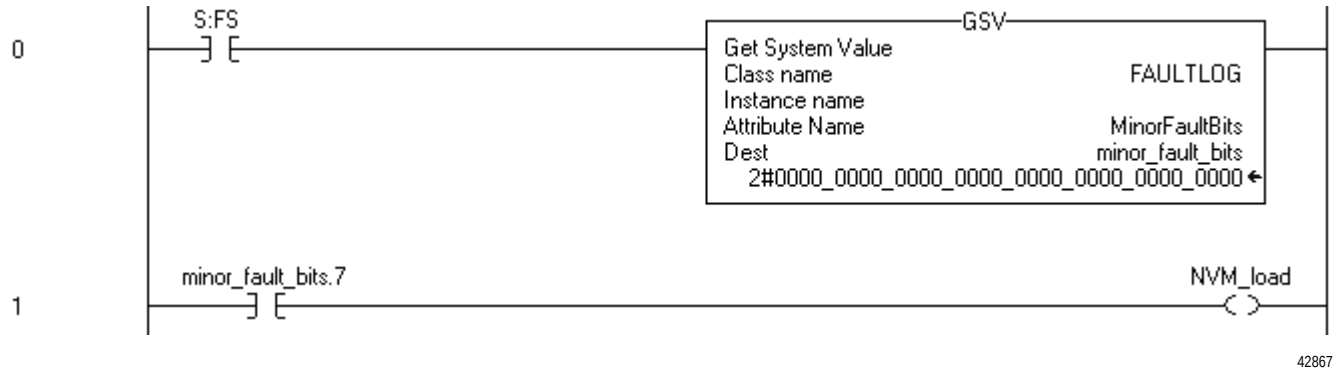
Check for a Load

When the controller loads a project from nonvolatile memory, it provides the following information:

- logs a minor fault (type 7, code 49)
- sets the FAULTLOG object, MinorFaultBits attribute, bit 7

If you want your project to flag that it loaded from nonvolatile memory, use this logic:

On the first scan of the project (*S:FS* is on), the GSV instruction gets the FAULTLOG object, MinorFaultBits attribute, and stores the value in *minor_fault_bits*. If bit 7 is on, the controller loaded the project from its nonvolatile memory.



Where:	Is:
`minor_fault_bits`	Tag that stores the FAULTLOG object, MinorFaultBits attribute. Data type is DINT.
`NVM_load`	Tag that indicates that the controller loaded the project from its nonvolatile memory.

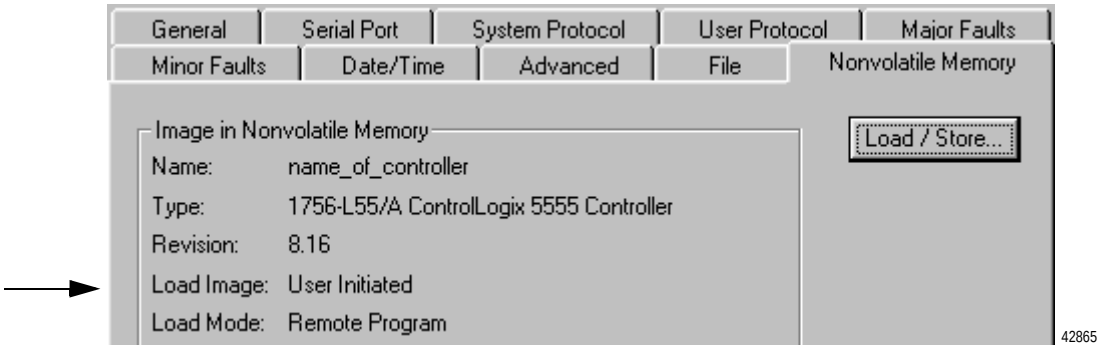
Clear Nonvolatile Memory

To remove a project from nonvolatile memory:

1. Go online with the controller.
2. On the Online toolbar, click the controller properties button.



3. Select the Nonvolatile Memory tab.



4. Is the Load Image option set to User Initiated?

If:	Then:
No	Go to “Change the Load Image Option” on 456.
Yes	Go to “Clear the Project from the Controller” on 457.

Change the Load Image Option

1. Click Load/Store.
2. In the Load Image drop-down list, select User Initiated.
3. Click <- Store.

   A dialog box asks you to confirm the store.
4. To store the project, choose Yes.

   A dialog box tells you that the store is in progress.
5. Click OK.
6. Wait until the OK LED on the front of the controller is steady green. This indicates that the store is finished.

Clear the Project from the Controller

1. Disconnect the battery from the controller.
2. Cycle the power to the chassis.
3. Re-connect the battery to the controller.

Store the Empty Image

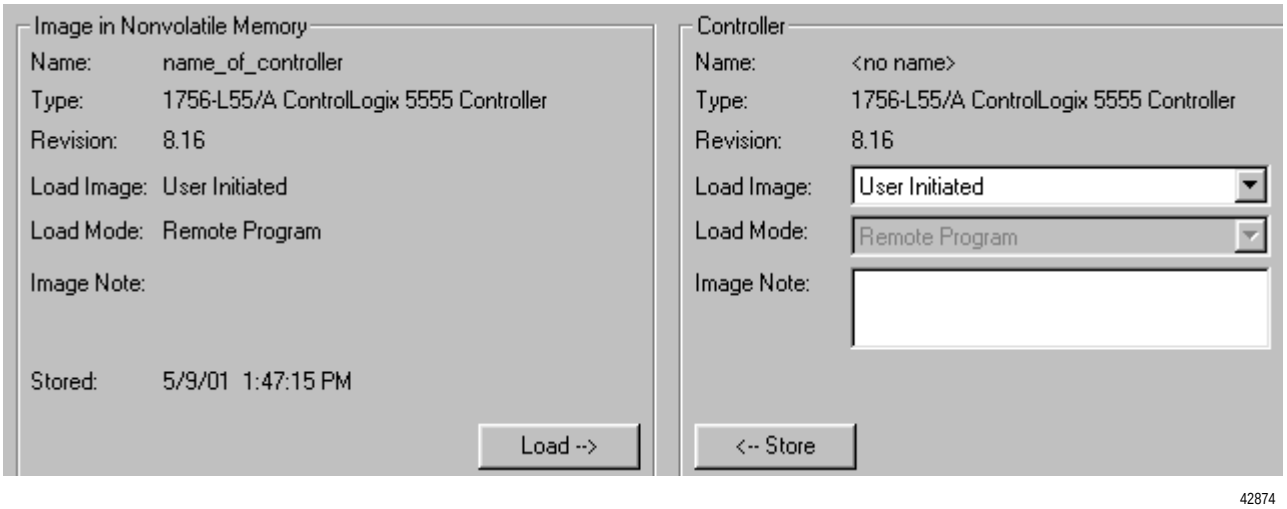
1. Go online with the controller.

   The Connected To Go Online dialog box opens.

2. Click the Nonvolatile Memory tab. and click Load/Store



3. Choose <– *Store*.



A dialog box asks you to confirm the store.

4. To store the project, choose *Yes*.

   During the store, the following events occur:

   - On the front of the controller, the OK LED displays this sequence: flashing green > red > green
   - RSLogix 5000 software goes offline.
   - A dialog box tells you that the store is in progress.

5. Choose *OK*.

When the store is finished, you remain offline.

Use a CompactFlash Reader

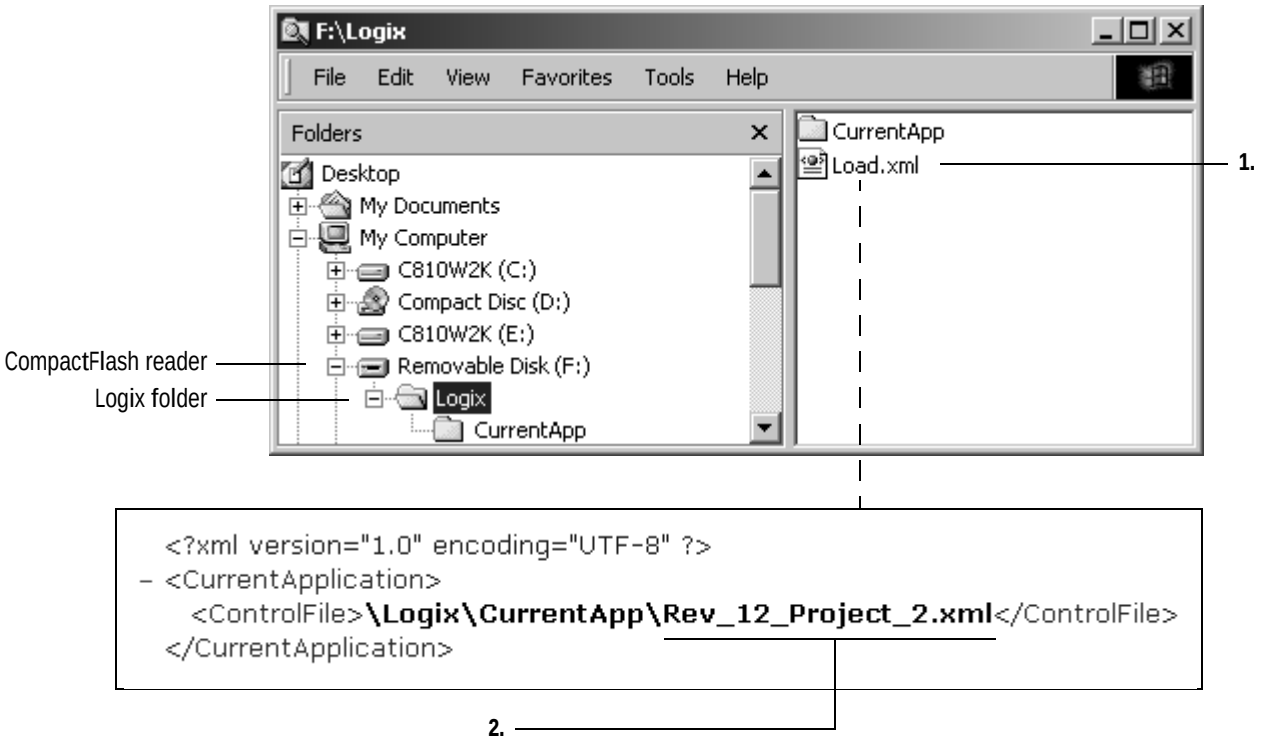
If the revision of the project or projects on your CompactFlash card are ≥ 12.0, then the card is formatted using the FAT16 file system.

A sample controller project that reads and writes a CompactFlash card is available with RSLogix 5000 Enterprise programming software. From RSlogix 5000 software, select Help → Vendor Sample Projects to display a list of available, sample projects.

Manually Change Which Project Loads from the CompactFlash Card

A CompactFlash card stores multiple projects. By default, the controller loads the project that you most recently stored, according to the load options of that project.

To assign a different project to load from the CompactFlash card, edit the Load.xml file on the card.



1. To change which project loads from the card, open *Load.xml*. Use a text editor to open the file.

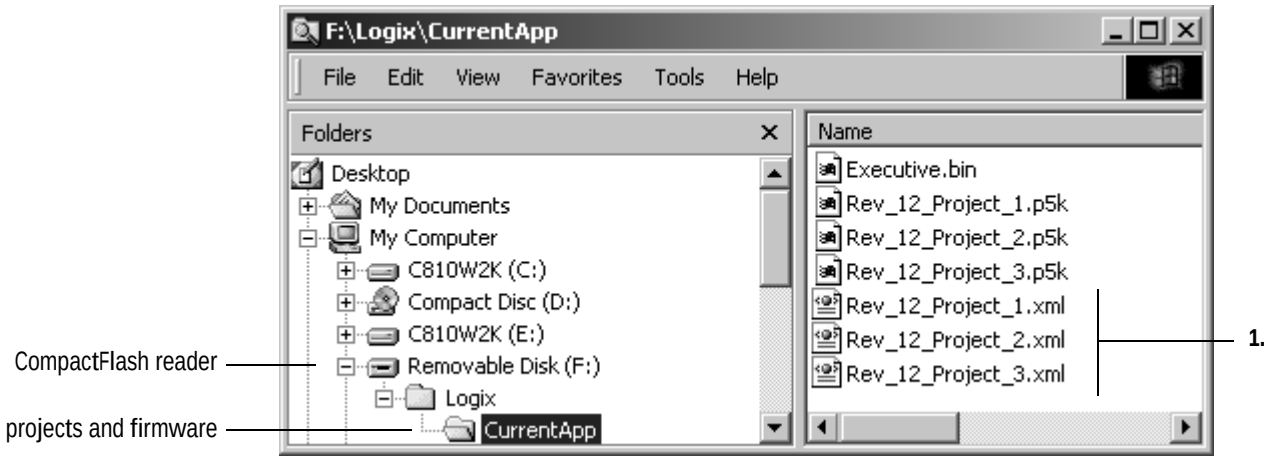
2. Edit the name of the project that you want to load.
   - Use the name of an XML file that is in the CurrentApp folder.
   - In the CurrentApp folder, a project is comprised of an XML file and a P5K file.

Manually Change the Load Parameters for a Project

When you store a project to nonvolatile memory, you define:

- when the project is to load (On Power Up, On Corrupt Memory, User Initiated)
- mode to which to set the controller (if the keyswitch is in REM and the load mode is not User Initiated)

To assign a different project to load from the CompactFlash card, edit the Load.xml file on the card.



1. To change the load parameters for a project, open the XML file with the same name as the project. Use a text editor to open the file.

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <Controller>
  - <ExecutiveLoadOption>
      <ExecFile>\Logix\CurrentApp\Executive.bin</ExecFile>
    </ExecutiveLoadOption>
  - <ProgramLoadOption>
2.    <ProgramLoadMode>CORRUPT_RAM</ProgramLoadMode>
      <LoadFile>\Logix\CurrentApp\Rev_12_Project_2.p5k</LoadFile>
    </ProgramLoadOption>
  - <ControllerModeOption>
3.    <ControllerMode>RUN</ControllerMode>
    </ControllerModeOption>
  </Controller>
```

2. Edit the Load Image option of the project.

If you want to set the Load Image option to:	Then enter:
On Power Up	ALWAYS
On Corrupt Memory	CORRUPT_RAM
User Initiated	USER_INITIATED

3. Edit the Load Mode option of the project (doesn't apply if the Load Image option is *User Initiated*).

If you want to set the Load Mode option to:	Then enter:
Program (Remote Only)	PROGRAM
Run (Remote Only)	RUN

Other Uses for a CompactFlash Card

Starting with revision 13, you can use the CompactFlash card to store data, in addition to controller projects. For example:

- A PanelView terminal changes tag values in a controller project. If power to the controller is lost (and the controller is not battery backed up), the program running in the controller, along with any values that were changed by the PanelView terminal will be lost. Use the CompactFlash file system and logic in the project to store tag values as they change. When the project reloads from the CompactFlash card, it can check the CompactFlash card for any saved tag values and reload those into the project.

- Store a collection of recipes on the CompactFlash card. When you need to change a recipe, program the controller to read data for the new recipe from a CompactFlash card.

- Program the controller to write data logs at specific time intervals.

You can also read and write CompactFlash cards using a CompactFlash card reader. This method writes tag values in binary. You can read the data using any text editor, but the data displays as the ASCII equivalent of the binary data.

For more information, see the sample projects available with RSLogix 5000 Enterprise programming software. From RSlogix 5000 software, select Help → Vendor Sample Projects to display a list of available, sample projects. You can also see the G82433612 Technical Note " Working with the CompactFlash File System on Logix5000 Controllers" from the MySupport Knowledgebase. Access this database selecting the Knowledgebase link from www.ab.com

Notes:

Determine Controller Memory Information

When to Use This Chapter

Use this chapter to get information about the memory of a Logix5000 controller.

To:	Page
Determine What Memory Information You Want	463
Estimate Memory Information Offline	464
View Run Time Memory Information	465
Write Logic to Get Memory Information	466

Determine What Memory Information You Want

Depending on your type of controller, the memory of the controller may be divided into several areas:

If you have this controller:	Then it stores this:	In this memory:
ControlLogix	I/O tags	I/O memory
	produced tags	
	consumed tags	
	communication via Message (MSG) instructions	
	communication with workstations	
	communication with polled (OPC/DDE) tags that use RSLinx software[1]	
	tags other than I/O, produced, or consumed tags	data and logic memory[2]
	Logic routines	
	communication with polled (OPC/DDE) tags that use RSLinx software[1]	
• CompactLogix • FlexLogix • DriveLogix • SoftLogix5800	These controllers do not divide their memory. They store all elements in one common memory area.	

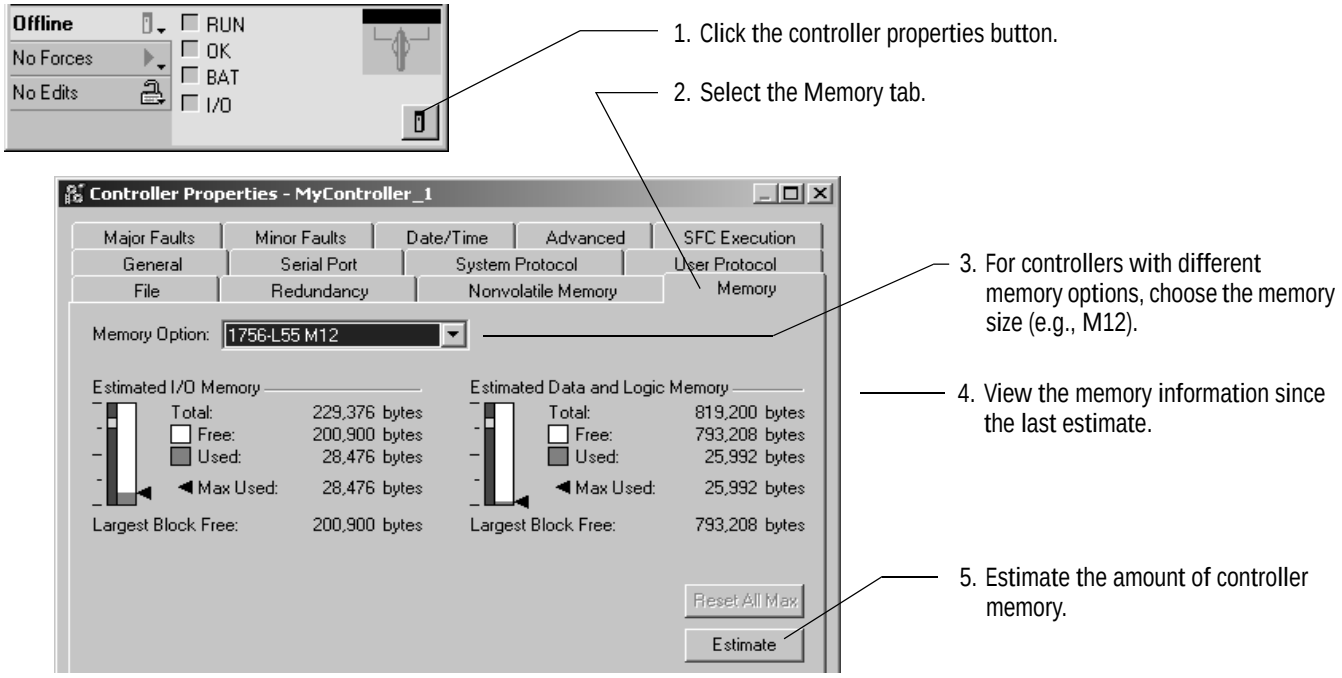
(1) To communicate with polled tags, the controller uses both I/O and data and logic memory.

(2) 1756-L55M16 controllers have an additional memory section for logic.

Estimate Memory Information Offline

To estimate how much controller memory your project requires, use the Memory tab of the controller properties dialog box. For each of the memory areas of your controller, it lets you estimate number of bytes of:

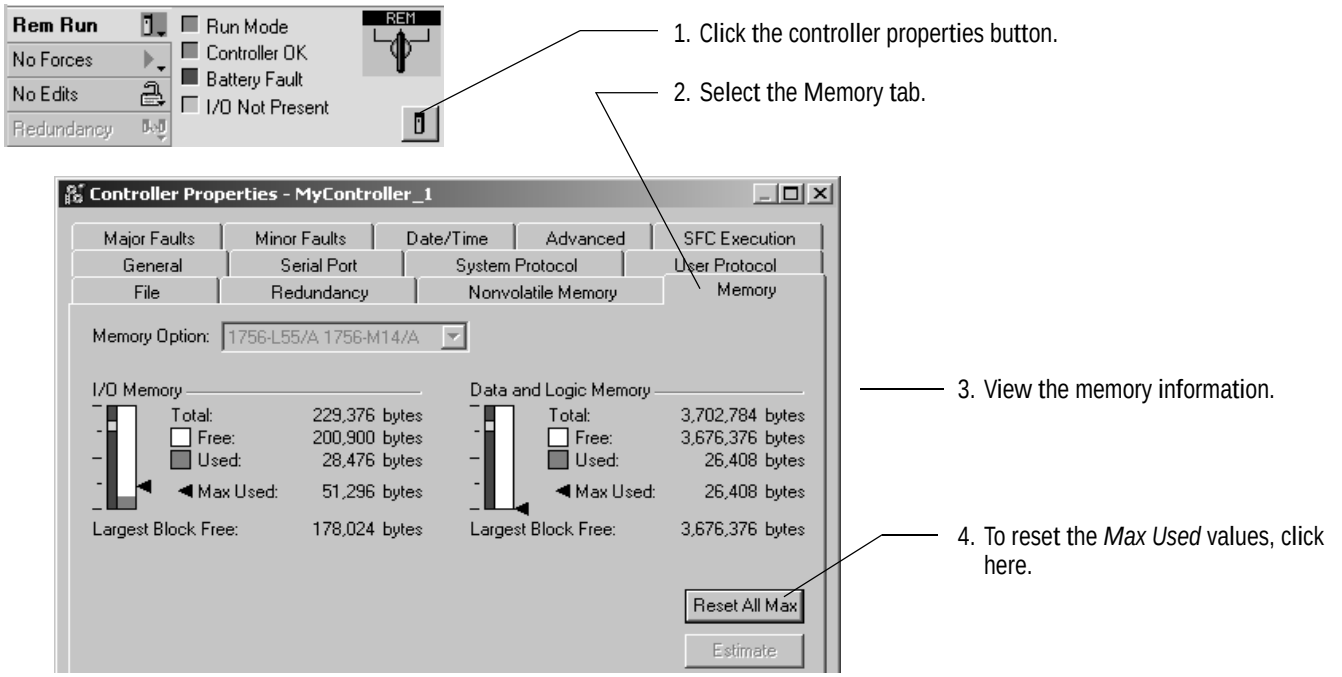
- free (unused) memory
- used memory
- largest free contiguous block of memory



View Run Time Memory Information

When online with a controller, the Memory tab shows the actual memory usage of the controller. While the controller is running, it uses additional memory for communication. The amount it needs varies depending on the state of the communication.

The Memory tab of the controller includes a Max Used entry for each type of memory. The Max Used values show the peak of memory usage as communications occur.



Write Logic to Get Memory Information

To use logic to get memory information for the controller:

- Get Memory Information from the Controller
- Choose the Memory Information That You Want
- Convert INTs to a DINT

Get Memory Information from the Controller

To get memory information from the controller, execute a Message (MSG) instruction that is configured as follows:

On this tab:	For this item:	Type or select:	Which means:
Configuration	Message Type	CIP Generic	Execute a Control and Information Protocol command.
	Service Type	Custom	Create a CIP Generic message that is not available in the drop-down list.
	Service Code	3	Read specific information about the controller (GetAttributeList service).
	Class	72	Get information from the user memory object.
	Instance	1	This object contains only 1 instance.
	Attribute	0	Null value
	Source Element	*source_array* of type SINT[12]	
		In this element:	**Enter:** **Which means:**
		source_array[0]	5 Get 5 attributes
		source_array[1]	0 Null value
		source_array[2]	1 Get free memory
		source_array[3]	0 Null value
		source_array[4]	2 Get total memory
		source_array[5]	0 Null value
		source_array[6]	5 Get largest contiguous block of additional free logic memory
		source_array[7]	0 Null value
		source_array[8]	6 Get largest contiguous block of free I/O memory
		source_array[9]	0 Null value
		source_array[10]	7 Get largest contiguous block of free data and logic memory
		source_array[11]	0 Null value
	Source Length	12	Write 12 bytes (12 SINTs).
	Destination	*INT_array* of type INT[29]	
Communication	Path	1, *slot_number_of_controller*	

Choose the Memory Information That You Want

The MSG instruction returns the following information to INT_array (destination tag of the MSG):

IMPORTANT

- The controller returns the values in number of 32-bit words. To see a value in bytes, multiple it by 4.
- If your controller does not divide its memory, then the values show up as I/O memory.
- For a 1756-L55M16 controller, the MSG instruction returns two values for each logic memory category. To determine the free or total logic memory of a 1756-L55M16 controller, add both values for the category.

If you want the:	Then copy these array elements:	Description:
amount of free I/O memory (32-bit words)	*INT_array*[3]	lower 16 bits of the 32 bit value
	INT_array[4]	upper 16 bits of the 32 bit value
amount of free data and logic memory (32-bit words)	*INT_array*[5]	lower 16 bits of the 32 bit value
	INT_array[6]	upper 16 bits of the 32 bit value
1756-L55M16 controllers only—amount of additional free logic memory (32-bit words)	*INT_array*[7]	lower 16 bits of the 32 bit value
	INT_array[8]	upper 16 bits of the 32 bit value
total size of I/O memory (32-bit words)	*INT_array*[11]	lower 16 bits of the 32 bit value
	INT_array[12]	upper 16 bits of the 32 bit value
total size of data and logic memory (32-bit words)	*INT_array*[13]	lower 16 bits of the 32 bit value
	INT_array[14]	upper 16 bits of the 32 bit value
1756-L55M16 controllers only—additional logic memory (32-bit words)	*INT_array*[15]	lower 16 bits of the 32 bit value
	INT_array[16]	upper 16 bits of the 32 bit value
1756-L55M16 controllers only—largest contiguous block of additional free logic memory (32-bit words)	*INT_array*[19]	lower 16 bits of the 32 bit value
	INT_array[20]	upper 16 bits of the 32 bit value
largest contiguous block of free I/O memory (32-bit words)	*INT_array*[23]	lower 16 bits of the 32 bit value
	INT_array[24]	upper 16 bits of the 32 bit value
largest contiguous block of free data and logic memory (32-bit words)	*INT_array*[27]	lower 16 bits of the 32 bit value
	INT_array[28]	upper 16 bits of the 32 bit value

Convert INTs to a DINT

The MSG instruction returns each memory value as two separate INTs.

- The first INT represents the lower 16 bits of the value.
- The second INT represents the upper 16 bits of the value.

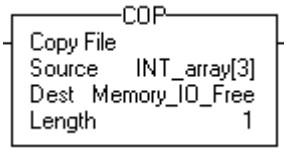
To convert the separate INTs into one usable value, use a Copy (COP) instruction, where:

In this operand:	Specify:	Which means:
Source	first INT of the 2 element pair (lower 16 bits)	Start with the lower 16 bits
Destination	DINT tag in which to store the 32-bit value	Copy the value to the DINT tag.
Length	1	Copy 1 times the number of bytes in the Destination data type. In this case, the instruction copies 4 bytes (32 bits), which combines the lower and upper 16 bits into one 32-bit value.

In the following example, the COP instruction produces the 32-bit value that represents the amount of free I/O memory, in 32-bit words.

EXAMPLE Convert INTs to a DINT

- Elements 3 of INT_array is the lower 16 bits of the amount of free I/O memory. Element 4 is the upper 16 bits.
- Memory_IO_Free is a DINT tag (32 bits) in which to store the value for the amount of free I/O memory.
- To copy all 32 bits, specify a Length of 1. This tells the instruction to copy 1 times the size of the Destination (32 bits). This copies both element 3 (16 bits) and element 4 (16 bits) and places the 32-bit result in Memory_IO_Free.



Manage Multiple Messages

When to Use This Appendix

This appendix describes how to use Ladder logic to send groups of Message (MSG) instructions in sequence. Use this appendix if you need to control the execution of a large number of MSGs.

- To be processed, each MSG instruction must enter the message queue.
- The queue holds 16 MSGs.
- If more than 16 MSGs are enabled at one time, there may not be room on the queue when a MSG is enabled.
- If this occurs, the MSG has to wait until there is room on the queue before the controller can process the MSG. On each subsequent scan of the MSG, it checks the queue to see if there is room.

The message manager logic in this appendix lets you control the number of MSGs that are enabled at one time and enable subsequent MSGs in sequence. In this way, MSGs enter and exit the queue in an ordered fashion and do not have to wait for room on the queue to become available.

How to Use this Appendix

In this appendix, the message manager logic sends three groups of MSGs.

- To make the example easier to follow, each groups contains only 2 MSGs.
- In your project, use more MSGs in each group, such as 5.
- Use as many groups as needed to include all your MSGs.

The Msg_Group tag controls the enabling of each MSG.

- The tag uses the DINT data type.
- Each bit of the tag corresponds to a group of MSGs.
- For example, Msg_Group.0 enables and disables the first group of MSGs (group 0).

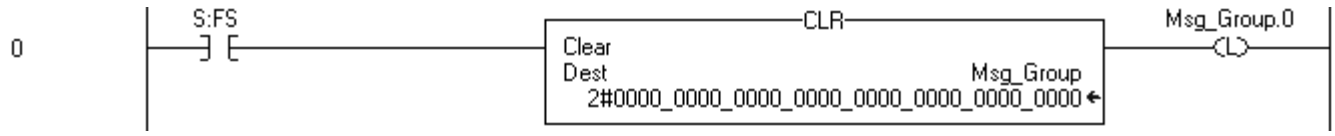
Message Manager Logic

Initialize the Logic

If S:FS = 1 (first scan), then initialize the MSGs:

Msg_Group = 0, which disables all the MSGs.

Msg_Group.0 =1, which enables the first group of MSGs.



Restart the Sequence, If Required

If the MSGs in group 2 (last group) are currently enabled (Msg_Group.2 = 1)

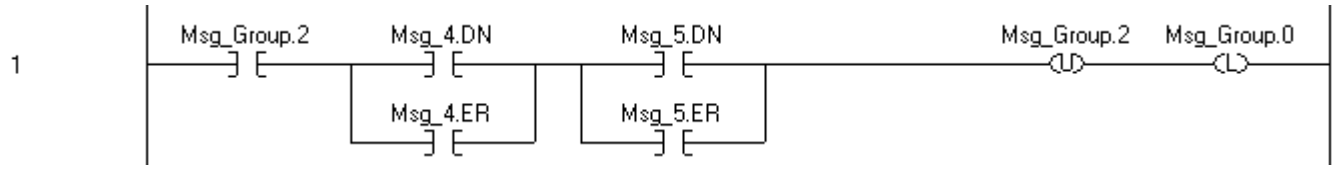
And Msg_4 is done or errored

And Msg_5 is done or errored

Then restart the sequence of MSGs with the first group:

Msg_Group.2 = 0. This disables the last group of MSGs.

Msg_Group.0 = 1. This enables the first group of MSGs.



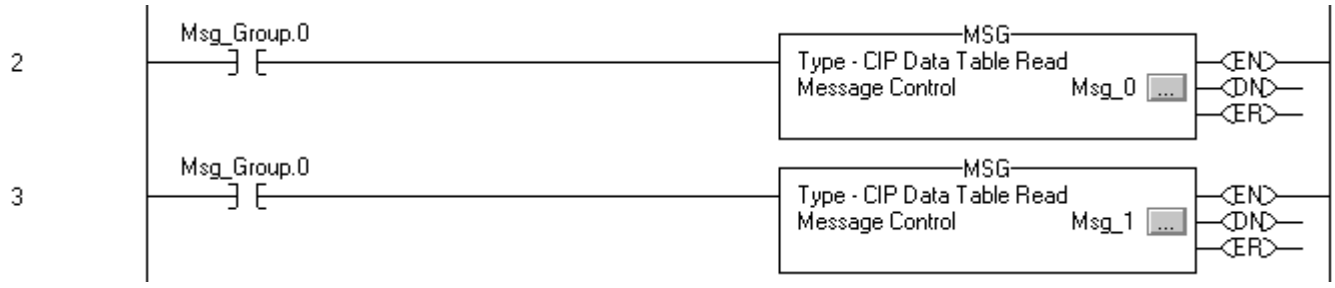
Send the First Group of MSGs

If Msg_Group.0 changes from 0 -> 1 then

Send Msg_0.

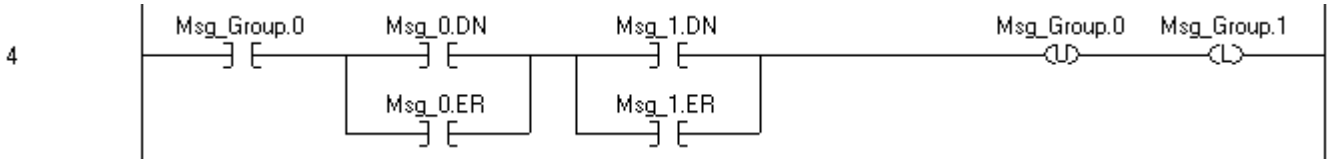
Send Msg_1.

Because a MSG instruction is a transitional instruction, it executes only when its rung-condition-in changes from false to true.



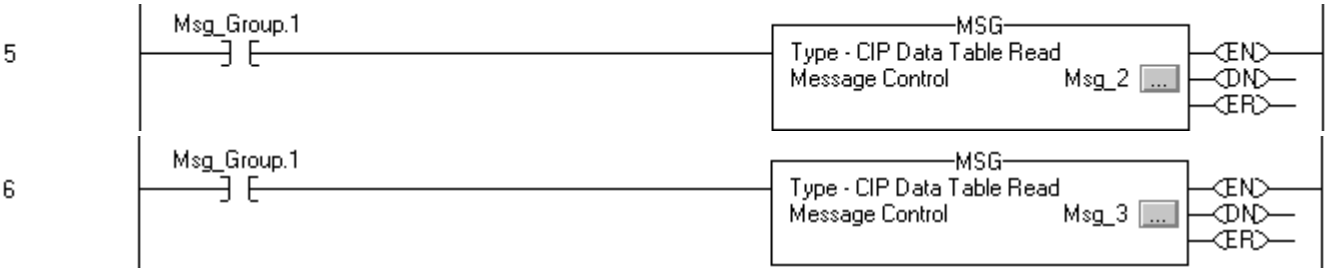
Enable the Next Group of MSGs

If the MSGs in group 0 are currently enabled (Msg_Group.0 = 1)
And Msg_0 is done or errored
And Msg_1 is done or errored
Then
Msg_Group.0 = 0. This disables the current group of MSGs.
Msg_Group.1 = 1. This enables the next group of MSGs.



Send the Next Group of MSGs

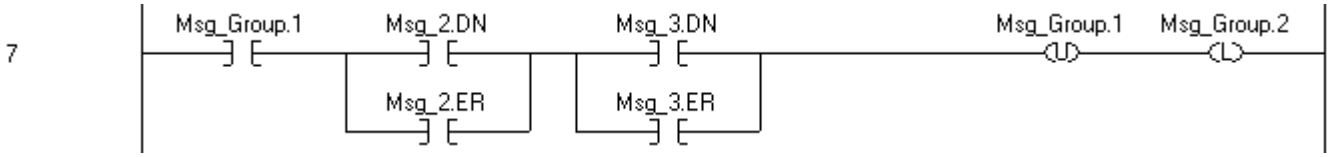
If Msg_Group.1 changes from 0 -> 1 then
Send Msg_2.
Send Msg_3.



Enable the Next Group of MSGs

If the MSGs in group 1 are currently enabled (Msg_Group.1 = 1)
And Msg_2 is done or errored
And Msg_3 is done or errored
Then

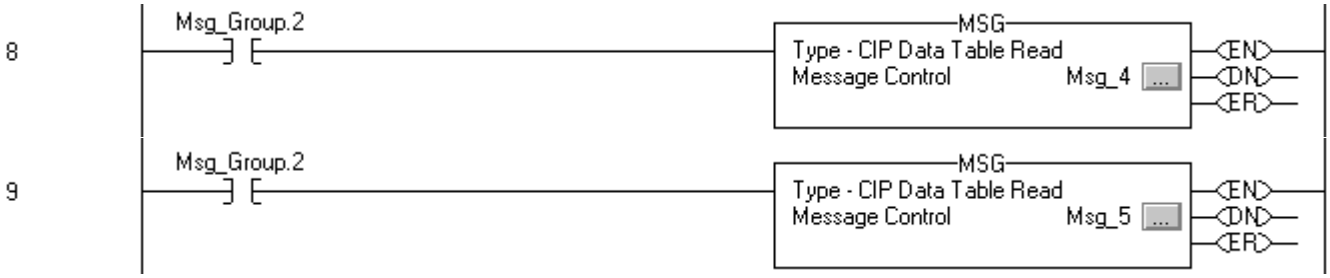
Msg_Group.1 = 0. This disables the current group of MSGs.
Msg_Group.2 = 1. This enables the next group of MSGs.



Send the Next Group of MSGs

If Msg_Group.1 changes from 0 -> 1 then

Send Msg_2.
Send Msg_3.



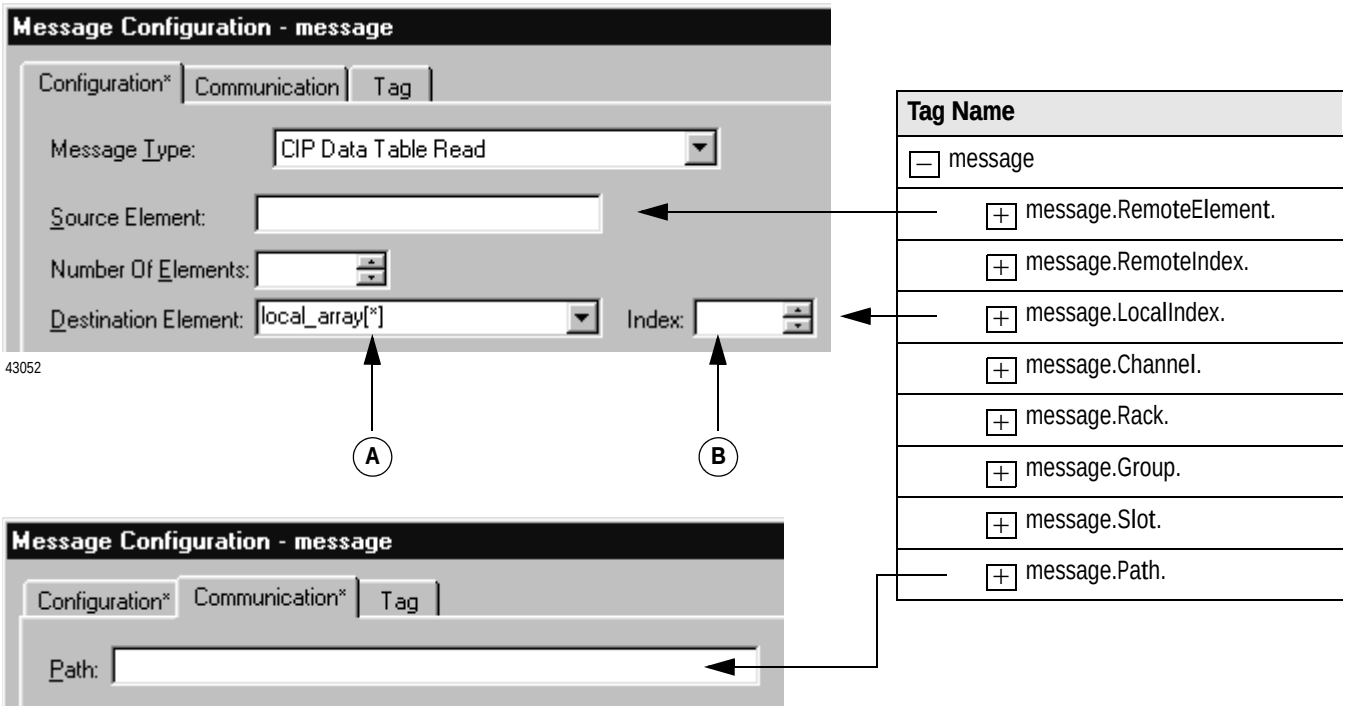
Send a Message to Multiple Controllers

When to Use This Appendix

Use this appendix to program a single message instruction to communicate with multiple controllers. To reconfigure a MSG instruction during runtime, write new values to the members of the MESSAGE data type.

IMPORTANT In the MESSAGE data type, the RemoteElement member stores the tag name or address of the data in the controller that receives the message.

If the message:	Then the RemoteElement is the:
reads data	Source Element
writes data	Destination Element



A. If you use an asterisk [*] to designate the element number of the array, the value in (B) provides the element number.

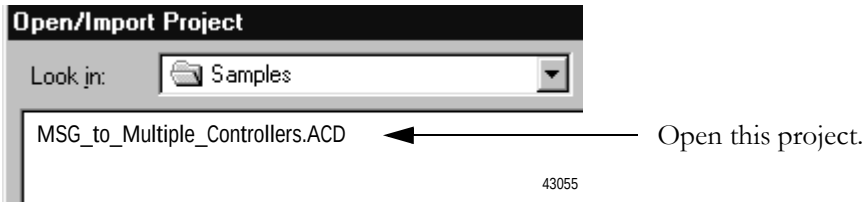
B. The Index box is only available when you use an asterisk [*] in the Source Element or Destination Element. The instruction substitutes the value of Index for the asterisk [*].

To send a message to multiple controllers:

- Set Up the I/O Configuration
- Define Your Source and Destination Elements
- Create the MESSAGE_CONFIGURATION Data Type
- Create the Configuration Array
- Get the Size of the Local Array
- Load the Message Properties for a Controller
- Configure the Message
- Step to the Next Controller
- Restart the Sequence

TIP

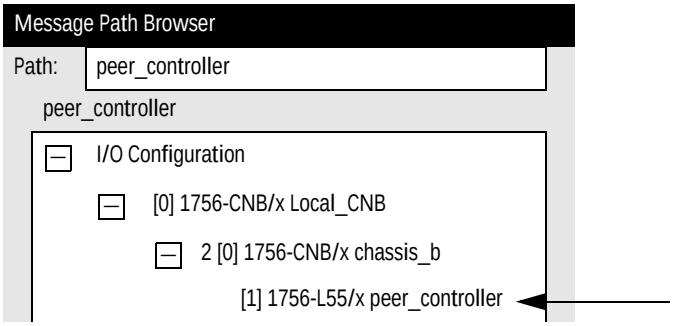
To copy the above components from a sample project, open the …\RSLogix 5000\Projects\Samples folder.



Set Up the I/O Configuration

Although not required, we recommend that you add the communication modules and remote controllers to the I/O configuration of the controller. This makes it easier to define the path to each remote controller.

For example, once you add the local communication module, the remote communication module, and the destination controller, the Browse button lets you select the destination.



Define Your Source and Destination Elements

In this procedure, an array stores the data that is read from or written to each remote controller. Each element in the array corresponds to a different remote controller.

1. Use the following worksheet to organize the tag names in the local and remote controllers:

Name of the remote controller:	Tag or address of the data in the remote controller:	Tag in this controller:
		local_array[0]
		local_array[1]
		local_array[2]
		local_array[3]

2. Create the local_array tag, which stores the data in this controller.

Tag Name	Type
local_array	*data_type* [*length*] where: *data_type* is the data type of the data that the message sends or receives, such as DINT, REAL, or STRING. *length* is the number of elements in the local array.

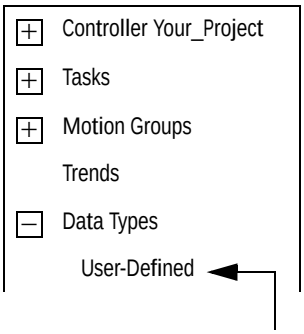
Create the MESSAGE_CONFIGURATION Data Type

In this procedure, you create a user-defined data type to store the configuration variables for the message to each controller.

- Some of the required members of the data type use a string data type.
- The default STRING data type stores 82 characters.
- If your paths or remote tag names or addresses use less than 82 characters, you have the option of creating a new string type that stores fewer characters. This lets you conserve memory.
- To create a new string type, choose File > New Component > String Type…
- If you create a new string type, use it in place of the STRING data type in this procedure.

To store the configuration variables for the message to each controller, create the following user-defined data type.

To create a new data type:



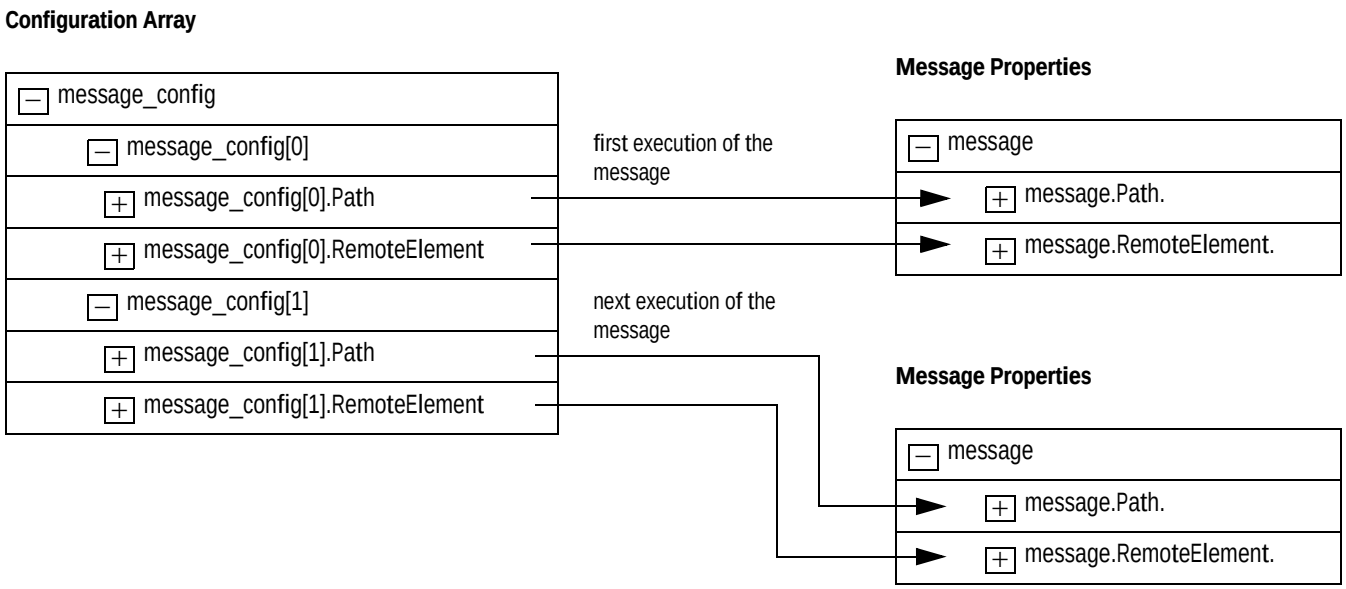
Right-click and choose New Data Type.

Data Type: MESSAGE_CONFIGURATION				
Name	MESSAGE_CONFIGURATION			
Description	Configuration properties for a message to another controller			
Members				
	Name	**Data Type**	**Style**	**Description**
	Path	STRING		
	RemoteElement	STRING		

Create the Configuration Array

In this procedure, you store the configuration properties for each controller in an array. Before each execution of the MSG instruction, your logic loads new properties into the instruction. This sends the message to a different controller.

Figure B.1 Load New Configuration Properties Into a MSG Instruction



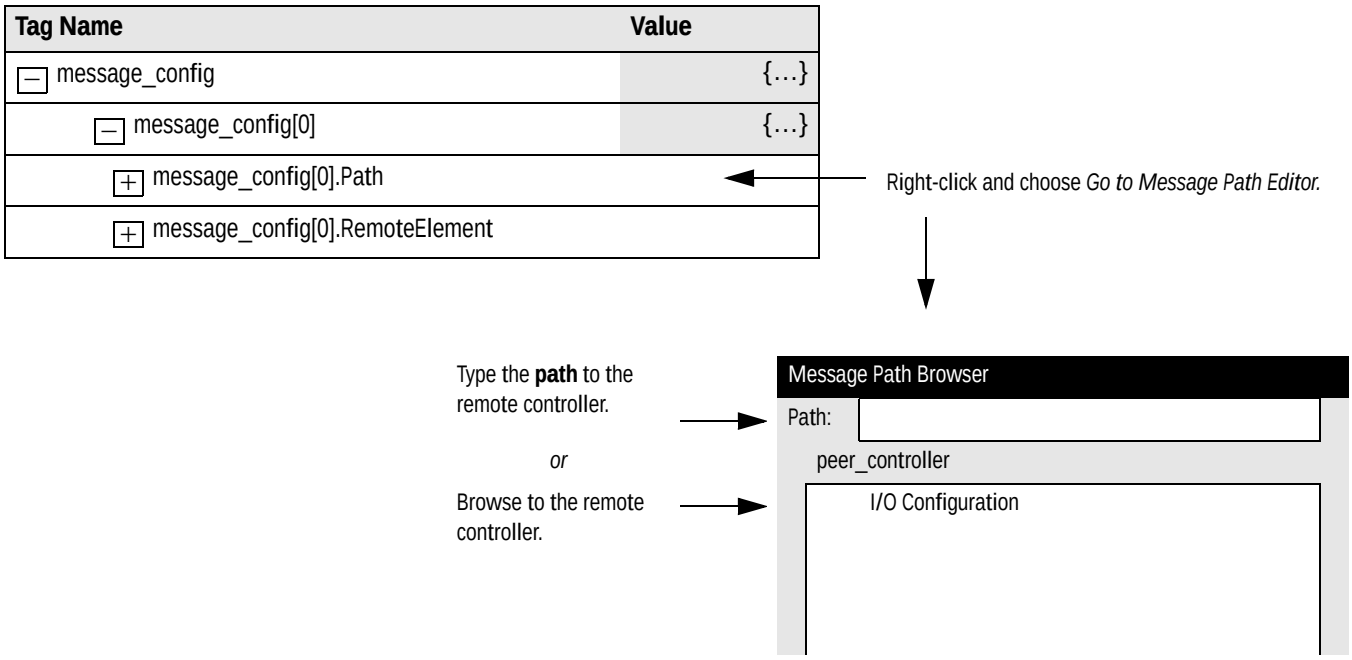
1. To store the configuration properties for the message, create the following array:

Tag Name	Type	Scope
message_config	MESSAGE_CONFIGURATION[*number*]	any

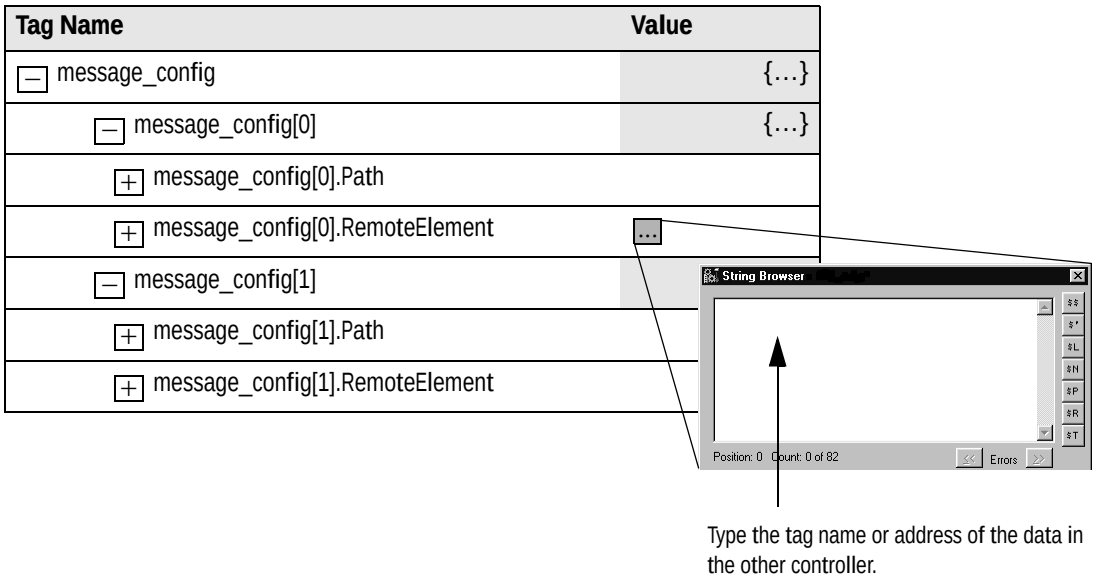
where:

number is the number of controllers to which to send the message.

2. Into the message_config array, enter the path to the first controller that receives the message.



3. Into the message_config array, enter the tag name or address of the data in the first controller to receive the message.



4. Enter the path and remote element for each additional controller:

Tag Name	Value
⊟ message_config	{...}
⊟ message_config[0]	{...}
⊞ message_config[0].Path	
⊞ message_config[0].RemoteElement	
⊟ message_config[1]	{...}
⊞ message_config[1].Path	
⊞ message_config[1].RemoteElement	

Get the Size of the Local Array

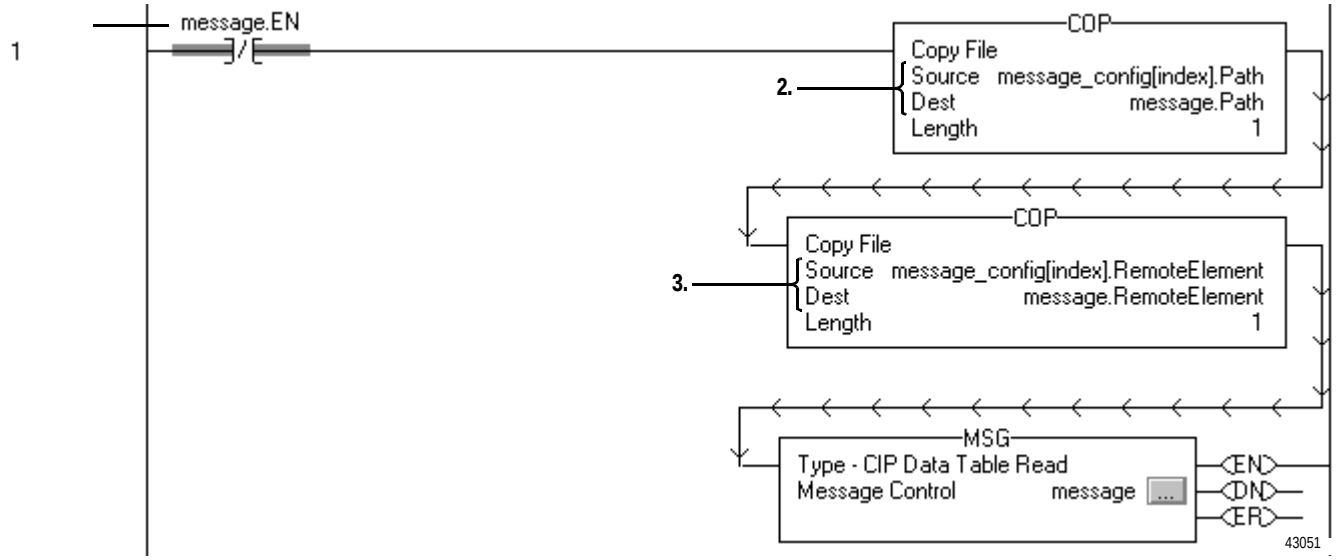


1. The SIZE instruction:

- counts the number of elements in local_array.
- counts the number of elements in Dimension 0 of the array. In this case, that is the only dimension.

2. Local_array_length (DINT) stores the size (number of elements) of local_array. This value tells a subsequent rung when the message has been sent to all the controllers and to start with the first controller again.

Load the Message Properties for a Controller



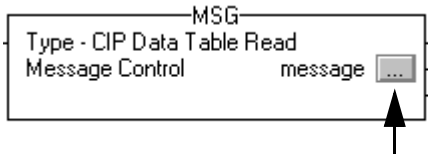
1. The XIO instruction conditions the rung to continuously send the message.

2. The first COP instruction loads the path for the message. The value of index determines which element the instruction loads from message_config. See Figure B.1 on page B-477.

   The instruction loads 1 element from message_config.

3. The second COP instruction loads the tag name or address of the data in the controller that receives the message. The value of index determines which element the instruction loads from message_config. See Figure B.1 on page B-477.

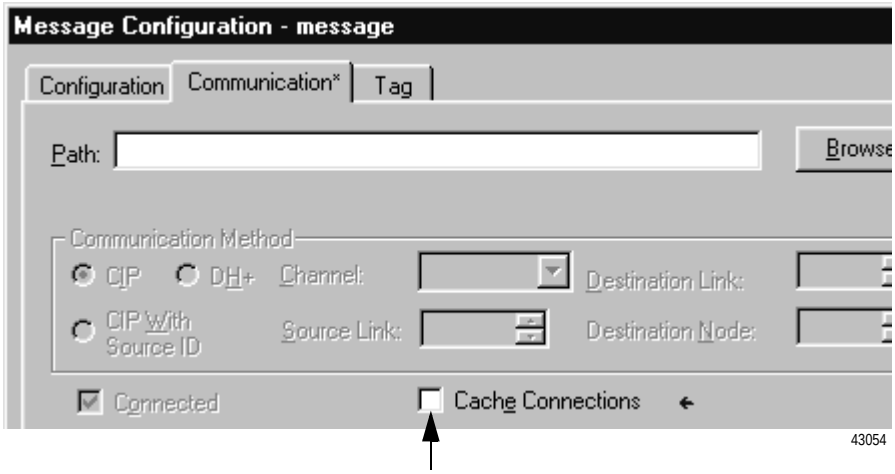
   The instruction loads 1 element from message_config.



Configure the Message

Although the logic controls the remote element and path for the message, there is initial configuration.

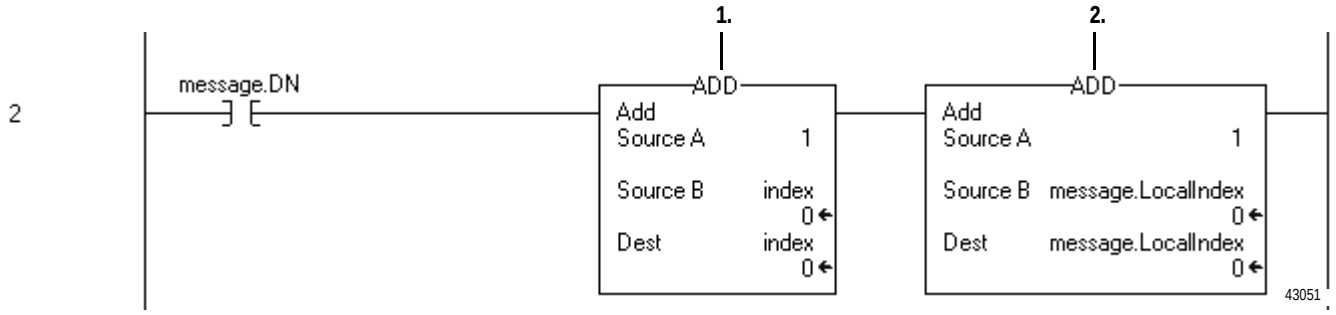
IMPORTANT



Clear the *Cache Connection* check box.

On this tab:	If you want to:	For this item:	Type or select:
Configuration	read (receive) data from the other controllers	Message Type	the read-type that corresponds to the other controllers
		Source Element	tag or address that contains the data in the first controller
		Number Of Elements	1
		Destination Tag	local_array[*]
		Index	0
	write (send) data to the other controllers	Message Type	the write-type that corresponds to other controllers
		Source Tag	local_array[*]
		Index	0
		Number Of Elements	1
		Destination Element	tag or address that contains the data in the first controller
Communication	→	Path	path to the first controller
		Cache Connections	Clear the Cache Connection check box. Since this procedure continuously changes the path of the message, it is more efficient to clear this check box.

Step to the Next Controller

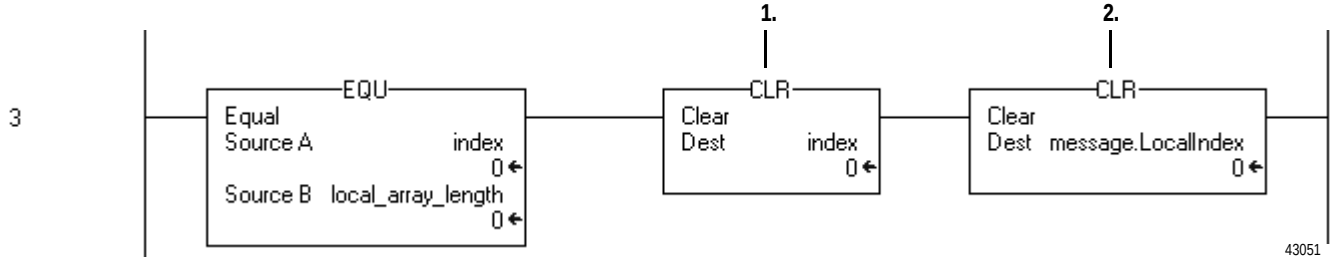


After the MSG instruction sends the message…

1. The first ADD instruction increments index. This lets the logic load the configuration properties for the next controller into the MSG instruction.

2. The second ADD instruction increments the LocalIndex member of the MSG instruction. This lets the logic load the value from the next controller into the next element of local_array.

Restart the Sequence



When index equal local_array_length, the controller has sent the message to all the other controllers.

1. The first CLR instruction sets index equal to 0. This lets the logic load the configuration properties for the first controller into the MSG instruction and start the sequence of messages again.

2. The second CLR instruction sets the LocalIndex member of the MSG instruction equal to 0. This lets the logic load the value from the first controller into the first element of local_array.

IEC61131-3 Compliance

Use This Appendix

For information about:	Page
Operating System	484
Data Definitions	484
Programming Languages	485
Instruction Set	486
IEC61131-3 Program Portability	486
IEC Compliance Tables	487

Introduction

The International Electrotechnical Commission (IEC) has developed a series of specifications for programmable controllers. These specifications are intended to promote international unification of equipment and programming languages for use in the controls industry. These standards provide the foundation for Logix5000 controllers and RSLogix 5000 programming software.

The IEC programmable controller specification is broken down into five separate parts each focusing on a different aspect of the control system:

- Part 1: General Information
- Part 2: Equipment and Requirements Test
- Part 3: Programming Languages
- Part 4: User Guidelines
- Part 5: Messaging Service Specification

The controls industry as a whole has focused on part 3 (IEC61131-3), Programming Languages, because it provides the cornerstone for implementing the other standards and provides the most significant end user benefit by reducing training cost. Because of this, only IEC61131-3 is addressed here.

The IEC61131-3 programming language specification addresses numerous aspects of programmable controller including the operating system execution, data definitions, programming languages, and instruction set. Components of the IEC61131-3 specification are categorized as required by the specification, optional or extensions. By so doing, the IEC61131-3 specification provides a minimum set of functionality that can be extended to meet end user application needs. The downside of this approach is that each programmable control system vendor may implement different components of the specification or provide different extensions.

Operating System

The preemptive, multitasking operating system (OS) of Logix5000 controllers complies with the IEC61131-3 definition. In IEC61131-3, the programmable controllers OS can contain zero or more tasks, that can execute one or more programs each containing one or more functions or routines. According to IEC61131-3, the number of each of these components is implementation dependent. Logix5000 controllers provide multiple tasks, each containing multiple programs and an unlimited number of functions or routines.

IEC61131-3 provides an option for creating different task execution classifications. Tasks may be configured as continuous, periodic, or event based. A continuous task does not need to be scheduled in that it will utilize any left over processing time when other tasks are dormant. Periodic tasks are scheduled to operate based on a reoccurring time period. The IEC61131-3 specification does not specify a time base for periodic task configuration. An IEC61131-3 event based task is triggered upon detection of the rising edge of a configured input. Logix5000 controllers support both continuous and periodic tasks. Additionally, the period for a periodic task is configurable starting as low as 1 millisecond (ms).

Data Definitions

The IEC61131-3 specification provides access to memory through the creation of named variables. IEC61131-3 names for variables consist of a minimum of six characters (RSLogix5000 programming software supports a minimum of 1 character) starting with an underscore "_" or an alpha character (A-Z), followed by one or more characters consisting of an underscore "_", alpha character (A-Z) or a number (0-9). Optionally, lower case alpha characters (a-z) can be supported as long as they are case insensitive (A = a, B = b, C = c …). Logix5000 controllers provide full compliance with this definition, support the lower case option, and extend the name to support up to 40 character names.

Data variables in IEC61131-3 may be defined such that they are accessible to all programs within a resource or controller, or limited access is provided only to the functions or routines within a single program. To pass data between multiple resources or controllers, access paths may be configured to define the location of the data within a system. Logix5000 controllers provide compliance by providing program scoped, controller scoped data and permits the configuration of access paths using produced/consumed data.

The memory interpretation of a variable within IEC61131-3 is defined through the use of either an elementary data type or an optional derived data type that is created from a group of multiple data types. Logix5000 controllers support the use of the BOOL (1 bit), SINT (8 bit integer), INT (16 bit integer), DINT (32 bit integer) and REAL (IEEE floating point number) elementary data types. Additionally, the optional derived data types are supported through the creation of user defined structures and arrays.

Programming Languages

The IEC61131-3 specification defines five (5) different programming languages and a set of common elements. All languages are defined as optional but at least one must be supported in order to claim compliance with the specification. The IEC61131-3 programming language components are defined as follows:

- Common Language Elements
- Common Graphical Elements
- Instruction List (IL) Language Elements
- Structured Text Language (ST) Elements
- Ladder Diagram (LD) Language Elements
- Sequential Function Chart (SFC) Language Elements
- Function Block Diagram (FBD) Language Elements

Logix5000 controllers and RSLogix5000 provide support for the common language elements and the Structured Text, Ladder Diagram, Sequential Function Chart, and Function Block Diagram language options. Additionally, the environment utilizes an ASCII import/export format based on the Structured Text language. The instruction set and program file exchange features are discussed in detail in the sections that follow.

Instruction Set

The instruction set specified by IEC61131-3 is entirely optional. The specification lists a limited set of instructions that if implemented must conform to the stated execution and visual representation. IEC61131-3 however, does not limit the instructions set to those listed within the specification. Each PLC vendor is free to implement additional functionality in the form of instructions over and above those listed by the specification. Examples of such extended instructions are those needed to perform diagnostics, PID loop control, motion control and data file manipulation. Because extended instructions are not defined by the IEC61131-3 specification, there is no guarantee that the implementation between different PLC vendors will be compatible. Thus utilization of these instructions may preclude the movement of logic between vendors.

Logix5000 controllers and RSLogix5000 provide a suite of instructions that execute as defined by the IEC61131-3 specification. The physical representation of these instructions maintain their look and feel with existing systems so as to reduce the training cost associated with working with the environment. In addition to the IEC61131-3 compliant instructions, a full range of instructions from existing products have been brought forward into the environment so that no functionality is lost.

IEC61131-3 Program Portability

One of the goals of end-users creating programs in an IEC61131-3 compliant environment is the movement or portability of programs between controllers developed by different vendors. This area is a weakness of IEC61131-3 because no file exchange format is defined by the specification. This means that if any program created in one vendor's environment will require manipulation to move it to another vendor's system.

In order to minimize the effort involved in performing cross-vendor portability, the RSLogix 5000 programming software for the controllers includes a full ASCII export and import utility. Additionally, the file format that is utilized by this tool is based on a hybrid of the IEC61131-3 Structured Text language definition. Controller operating system and data definitions follow the appropriate IEC61131-3 formats. Extensions were implemented in order to convert Ladder Diagram logic into ASCII text since this is not defined by IEC61131-3.

For more information on the ASCII export and import utility of RSLogix 5000 programming software, see the *Logix5000 Controllers Import/Export Reference Manual*, publication 1756-RM084.

IEC Compliance Tables

Logix5000 controllers and RSLogix5000 comply with the requirements of IEC61131-3 for the following language features:

Table Number:(1)	Feature Number:	Feature Description:	Extensions and Implementation Notes:
1	2	Lower case letters	none
1	3a	Number sign (#)	Used for immediate value data type designation
1	4a	Dollar sign ($)	Used for description and string control character
1	6a	Subscript delimiters ([ ])	Array subscripts
2	1	Identifiers using upper case and numbers	Task, program, routine, structure and tag names
2	2	Identifiers using upper case, numbers, and embedded underlines	Task, program, routine, structure and tag names
2	3	Identifiers using upper and lower case, numbers and embedded underlines	Task, program, routine, structure and tag names
3	1	Comments	ST Comments, also support /* Comment */, and // End of line comments.
4	1	Integer literal	12, 0, -12
4	2	Real literal	12.5, -12.5
4	3	Real literal with exponents	-1.34E-12, 1.234E6
4	4	Base 2 literal	2#0101_0101
4	5	Base 8 literal	8#377
4	6	Base 16 literal	16#FFE0
4	7	Boolean zero and one	0, 1
5	1A	Empty String ''	Descriptions, and String Editor
5	1B	String of length one containing a character 'A'	Descriptions, and String Editor
5	1C	String of length one containing a space ' '	Descriptions, and String Editor
5	1D	String of length one containing a single quote character '$''	Descriptions, and String Editor
5	1E	String of length one containing a double quote character '"'	Descriptions, and String Editor
5	1F	String of length two containing CR and LF characters	Descriptions, and String Editor
5	1G	String of length one containing the LF character '$0A'	Descriptions, and String Editor
5	1H	String of length 5 which would print as "$1.00" using '$$1.00'	Descriptions, and String Editor
5	1I	Equivalent strings of length two 'AE', and '$C4$CB'	Descriptions, and String Editor
6	2	String dollar sign '$$'	Descriptions, and String Editor
6	3	String single quote '$''	Descriptions, and String Editor
6	4	String Line Feed '$L' or '$l'	Descriptions, and String Editor

Table Number:(1)	Feature Number:	Feature Description:	Extensions and Implementation Notes:
6	5	String New-line '$N' or '$n'	Descriptions, and String Editor
6	6	String From Feed (page) '$P' or '$p'	Descriptions, and String Editor
6	7	String Carriage return '$R' or '$r'	Descriptions, and String Editor
6	8	String Tab '$T' or '$t'	Descriptions, and String Editor
6	9	String double quote $"	Descriptions, and String Editor
10	1	BOOL Data Type	Tag variable definition
10	2	SINT Data Type	Tag variable definition
10	3	INT Data Type	Tag variable definition
10	4	DINT Data Type	Tag variable definition
10	10	REAL Data Type	Tag variable definition
10	12	Time	Tag variable definition, TIMER Structure
10	16	STRING data type	8 Bits
11	1	Data type Hierarchy	none
12	1	Direct Derivation from elementary types	User Defined data type structures
12	4	Array data types	Tag variable definition
12	5	Structured Data types	User defined data type structures
13	1	BOOL, SINT, INT, DINT initial value of 0	Tag variable definition
13	4	REAL, LREAL initial value of 0.0	Tag variable definition
13	5	Time initial value of T#0s	Tag variable definition, reset (RES) instruction
13	9	Empty String ''	Descriptions and Strings
14	1	Initialization of directly derived types	Import/export
14	4	Initialization of array data types	Import/export
14	5	Initialization of structured type elements	Import/export
14	6	Initialization of derived structured data types	Import/export
19a	2a	Textual invocation, non-formal	Available in ST
20	1	Use of EN and ENO	Function present in LD but not labeled. Available in FBD.
20	2	Usage without EN and ENO	Available in FBD
20	3	Usage with EN and without ENO	Available in FBD
20	4	Usage without EN and with ENO	Available in FBD
21	1	Overloaded functions ADD(INT, DINT) or ADD(DINT, REAL)	All overloaded types that are supported are documented with each instruction
22	1	_TO_ conversion function	RAD, DEG instructions Radians to/from Decimal. String numeric conversion STOD, STOR, RTOS, DTOS. Others not needed because of instruction overloading
22	2	Truncate conversion function	TRN instruction in LD and TRUNC function in ST
22	3	BCD to INT Convert	FRD instruction in LD

Table Number:(1)	Feature Number:	Feature Description:	Extensions and Implementation Notes:
22	4	INT to BCD Convert	TOD instruction in LD
23	1	Absolute value	ABS instruction
23	2	Square root	SQR instruction in LD and FBD and SQRT function in ST.
23	3	Natural log	LN instruction
23	4	Log base 10	LOG instruction
23	6	Sine in radians	SIN instruction / function
23	7	Cosine in radians	COS instruction / function
23	8	Tangent in radians	TAN instruction / function
23	9	Principal arc sine	ASN instruction in LD and FBD, and ASIN function in ST
23	10	Principal arc cosine	ACS instruction in LD and FBD, and ACOS function in ST
23	11	Principal arc tangent	ATN instruction in LD and FBD, and ATAN function in ST
24	12	Arithmetic add	ADD instruction in LD and FBD, and + in ST.
24	13	Arithmetic multiplication	MUL instruction in LD and FBD, and * in ST.
24	14	Arithmetic subtraction	SUB instruction in LD and FBD, and - in ST.
24	15	Arithmetic divide	DIV instruction in LD and FBD, and / in ST.
24	16	Modulo	MOD instruction LD and ST
24	17	Exponentiation	XPY instruction in LD and FBD, and ** in ST.
24	18	Value move	MOV instruction in LD, and := in ST.
25	1	Bit shift left	Functionality contained in BSL instruction in LD for shift of 1
25	2	Bit shift right	Functionality contained in BSR instruction in LD for shift of 1
25	3	Bit rotate left	Functionality contained in BSL instruction in LD for shift of 1
25	4	Bit rotate right	Functionality contained in BSR instruction in LD for shift of 1
26	5	AND	BAND instruction in FBD, and "&" operator in ST
26	6	OR	BOR instruction in FBD
26	7	XOR	BXOR instruction in FBD
26	8	NOT	BNOT instruction in FBD
27	1	SELECT	SEL instruction in FBD
27	2a	Maximum select MAX	Functionality contained in ESEL instruction in FBD and ST
27	2b	Minimum select MIN	Functionality contained in ESEL instruction in FBD and ST

Table Number:(1)	Feature Number:	Feature Description:	Extensions and Implementation Notes:
27	3	High/Low limit LIMIT	HLL instruction in FBD and ST
27	4	Multiplexer MUX	MUX instruction in FBD
28	5	Comparison greater-than	GRT instruction in LD and FBD, and > in ST.
28	6	Comparison greater-than or equal	GRE instruction in LD and FBD, and >= in ST.
28	7	Comparison equal	EQU instruction in LD and FBD, and = in ST.
28	8	Comparison less-than	LES instruction in LD and FBD, and < in ST.
28	9	Comparison less-than or equal	LEQ instruction in LD and FBD, and <= in ST.
28	10	Comparison not equal	NEQ instruction in LD and FBD, and <> in ST.
29	1	String length LEN	Contained as Parameter of STRING data type
29	4	Middle string MID	MID instruction in LD and ST
29	5	String concatenation CONCAT	CONCAT instruction in LD and ST
29	6	String insert INSERT	INSERT instruction in LD and ST
29	7	String delete DELETE	DELETE instruction in LD and ST
29	9	Find string FIND	FIND instruction in LD and ST
32	1	Input read	FBD and ST
32	2	Input write	FBD and ST
32	3	Output read	FBD and ST
32	4	Output write	FBD and ST
34	1	Bistable set dominant	SETD instruction in FBD and ST
34	2	Bistable reset dominant	RESD instruction in FBD and ST
35	1	Rising edge detector	OSR instruction in LD and OSRI instruction in FBD and ST
35	2	Falling edge detector	OSF instruction in LD and OSFI instruction in FBD and ST
36	1b	Up-counter	Functionality contained in CTU and RES instructions in LD and in CTUD instruction in FBD and ST
37	2a	On-delay timer	Functionality contained in TON instruction in LD and TONR instruction in FBD and ST
37	3a	Off-delay timer	Functionality contained in TOF instruction in LD and TOFR instruction in FBD and ST
38	2	On-delay timing	Functionality contained in TON instruction in LD and TONR instruction in FBD and ST
38	3	Off-delay timing	Functionality contained in TOF instruction in LD and TOFR instruction in FBD and ST
40	1a	SFC Step	
40	1b	SFC initial Step	
40	2a	SFC Step Textual	Import/export, step name is specified using the format "Operand := step_name"

Table Number:(1)	Feature Number:	Feature Description:	Extensions and Implementation Notes:
40	2b	SFC initial Step textual	Import/export, uses "InitialStep" Parameter and step name is specified using the format "Operand := step_name"
40	3a	SFC Step Flag general form	Step backing tag
40	4	Step elapsed time general form	Step backing tag
41	1	Transition using ST	
41	5	Transition textual form	Import/export with different formatting
41	7	Transition Name	Transition Backing Tag
41	7a	Transition Set by LD	Transition Backing Tag
41	7b	Transition Set by FBD	Transition Backing Tag
41	7d	Transition Set by ST	Transition Backing Tag
42	1	Action Boolean	Action Backing tag
42	3s	Action textual representation	Import/export
43	1	Step Action association	
43	2	Step with Concatenated Actions	
43	3	Textual Step body	Import/export with different formatting
43	4	Action Body Field	Embedded ST
44	1	Action Block Qualifier	
44	2	Action Block Name	
44	3	Action Indicator Tag	Extended this to support DINT, INT, SINT, or REAL in addition to BOOL
44	5	Action using ST	Supports both embedded ST and JSR to ST routine
44	6	Action using LD	Using JSR to LD routine
44	7	Action using FBD	Using JSR to FBD Routine
45	1	Action Qualifier None	Default is N when none is explicitly entered
45	2	Action Qualifier N - Non-stored	
45	3	Action Qualifier R - Reset	
45	4	Action Qualifier S - Set / Stored	
45	5	Action Qualifier L - Time Limited	
45	6	Action Qualifier D - Time Delayed	
45	7	Action Qualifier P - Pulse	
45	8	Action Qualifier SD - Stored and Time Delayed	
45	9	Action Qualifier DS - Delayed and Stored	
45	10	Action Qualifier SL - Stored and time limited	
45	11	Action Qualifier P1 - Pulse Rising Edge	
45	12	Action Qualifier P0 - Pulse Falling Edge	

Table Number:(1)	Feature Number:	Feature Description:	Extensions and Implementation Notes:
45a	1	Action Control	
45a	2	Action Control	
46	1	SFC Single Sequence	
46	2a	SFC Divergence of sequence selection	Use of line connections vs. asterisk
46	2b	SFC Divergence of sequence selection with execution order.	
46	3	SFC Convergence of sequence selection	
46	4a	SFC Simultaneous sequence divergence	
46	4b	SFC Simultaneous sequence convergence	
46	5a, b, c	SFC Sequence Skip	
46	6a, b, c	SFC Sequence Loop	
46	7	SFC Loop directional arrows	When wire is hidden
47	1	SFC Graphical representation	
47	4	SFC Graphical representation	
48	1	SFC Minimal Step Compliance Requirements	Refer to notes on individual tables above.
48	2	SFC Minimal Transition Compliance Requirements	Refer to notes on individual tables above.
48	3	SFC Minimal Action Compliance Requirements	Refer to notes on individual tables above.
48	4	SFC Minimal Action Body Compliance Requirements	Refer to notes on individual tables above.
48	5	SFC Minimal Action Qualifier Compliance Requirements	Refer to notes on individual tables above.
48	6	SFC Minimal Branch Compliance Requirements	Refer to notes on individual tables above.
48	7	SFC Minimal Block Connection Compliance Requirements	Refer to notes on individual tables above.
55	1	ST Parenthesization (expression)	
55	2	ST Function Evaluation	Using non-formal form of invocation for built in functions. JSR used within ST language to call user developed code.
55	3	ST Exponentiation **	
55	4	ST Negation -	
55	5	ST Negation NOT	
55	6	ST Multiply *	
55	7	ST Divide /	
55	8	ST Modulo MOD	
55	9	ST Add +	
55	10	ST Subtract -	
55	11	ST Comparison <, >, <=, >=	

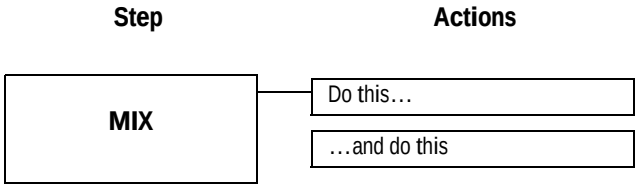
Table Number:(1)	Feature Number:	Feature Description:	Extensions and Implementation Notes:
55	12	ST Equality =	
55	13	ST Inequality <>	
55	14	ST Boolean AND as &	
55	15	ST Boolean AND	
55	16	ST Boolean XOR	
55	17	ST Boolean OR	
56	1	ST Assignment :=	
56	2	ST Function Block invocation	
56	3	ST RETURN	RET( ) with multiple Parameters
56	4	ST IF / ELSIF / ELSE/ END_IF	
56	5	ST CASE OF / ELSE / END_CASE	
56	6	ST FOR / END_FOR	
56	7	ST WHILE DO / END_WHILE	
56	8	ST REPEATE / UNTIL / END_REPEAT	
56	9	ST EXIT	
56	10	ST Empty Statement ;	
57	1, 2	Horizontal line	LD editor, FBD editor
57	3, 4	Vertical line	LD editor, FBD editor
57	5, 6	Horizontal / Vertical connection	LD editor, FBD editor
57	7, 8	Line crossings without connection	FBD editor
57	9, 10	Connection and non-connection corners	LD editor, FBD editor
57	11, 12	Blocks with connections	LD editor, FBD editor
57	13,14	Connectors	FBD editor
58	2	Unconditional jump	JMP instruction in LD
58	3	Jump target	LBL instruction in LD
58	4	Conditional jump	JMP instruction in LD
58	5	Conditional return	RET instruction in LD
58	8	Unconditional return	RET instruction in LD
59	1	Left hand power rail	LD editor
59	2	Right hand power rail	LD editor
60	1	Horizontal link	LD editor
60	2	Vertical link	LD editor
61	1, 2	Normally open contact --| |--	XIC instruction in LD
61	3, 4	Normally close contact --| / |--	XIO instruction in LD
61	5, 6	Positive transition sensing contact -| P |-	ONS instruction in LD
62	1	Coil --( )--	OTE instruction in LD

Table Number:(1)	Feature Number:	Feature Description:	Extensions and Implementation Notes:
62	3	Set (latch) coil	Functionality contained in OTL instruction in LD
62	4	Reset (unlatch) coil	Functionality contained in OTU instruction in LD
62	8	Positive transition sensing coil	OSR instruction in LD
62	9	Negative transition sensing coil	OSF instruction in LD

(1) Table associated with languages other than Structured Text, sequential function chart, Ladder diagram and Function Block diagram have been skipped.

A

action In a sequential function chart (SFC), an action represents a functional division of a step. Several actions make up a step. Each action performs a specific function, such as controlling a motor, opening a valve, or placing a group of devices in a specific mode.



Each action includes a qualifier. When a step is active (executing) the qualifier determines when the action starts and stops.

See **Sequential Function Chart**, **step**, **qualifier**.

Add-On Instruction Add-On Instructions are custom instructions that are added and available for use in a RSLogix 5000 project. They are similar to built-in instructions or subroutines that can be called by programming but are an entity to themselves so they become like any other RSLogix 5000 built-in instruction.

alias tag A tag that references another tag. An alias tag can refer to another alias tag or a base tag. An alias tag can also refer to a component of another tag by referencing a member of a structure, an array element, or a bit within a tag or member. See **base tag**.

ASCII A 7-bit code (with an optional parity bit) that is used to represent alphanumerical characters, punctuation marks, and control-code characters. For a list of ASCII codes, see the back cover of this manual.

asynchronous Actions that occur independent of each other and lack a regular pattern. In Logix5000 controllers, I/O values update asynchronous to the execution of logic:

- Programs within a task access input and output data directly from controller-scoped memory.
- Logic within any task can modify controller-scoped data.
- Data and I/O values are asynchronous and can change during the course of a task's execution.
- An input value referenced at the beginning of a task's execution can be different when referenced later.

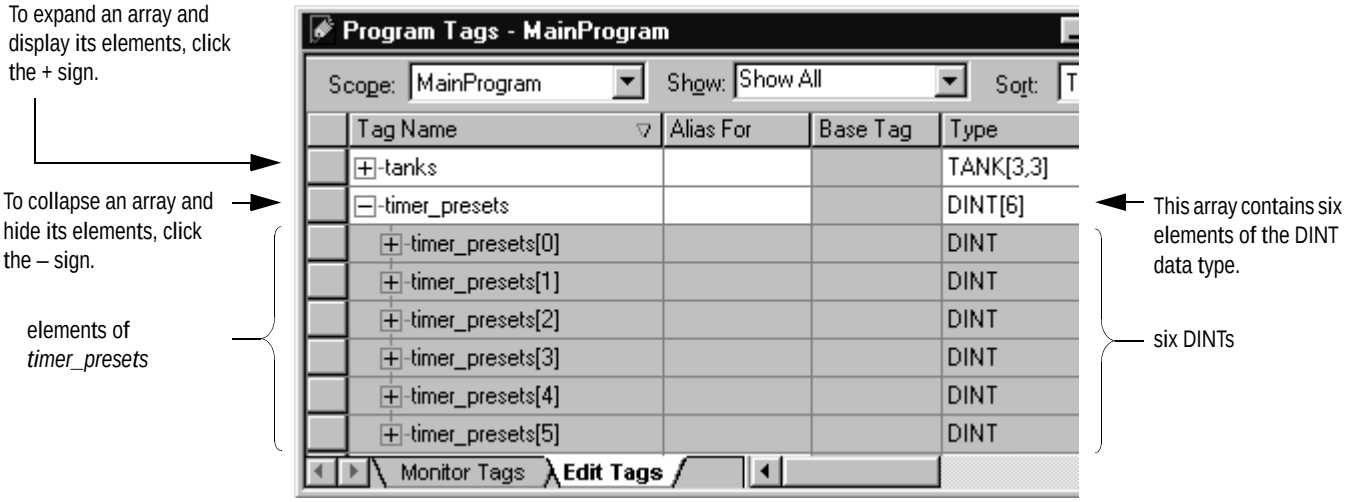
ATTENTION



Take care to ensure that data memory contains the appropriate values throughout a task's execution. You can duplicate or buffer data at the beginning of the scan to provide reference values for your logic.

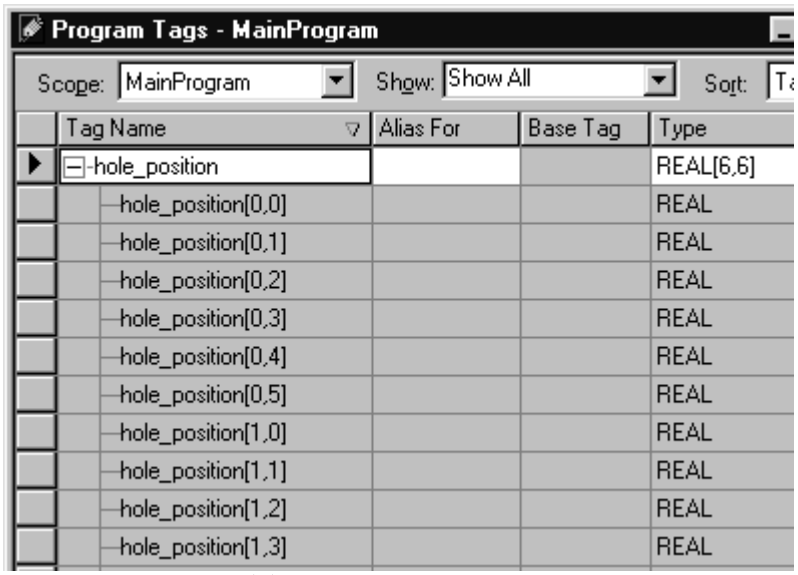
array An array lets you group data (of the same data type) under a common name.

- An array is similar to a file.
- A subscript (s) identifies each individual **element** within the array.
- A subscript starts at 0 and extends to the number of elements minus 1 (zero based).



- An array tag occupies a contiguous block of memory in the controller, each element in sequence.
- You can use array and sequencer instructions to manipulate or index through the elements of an array

- An array can have as many as three dimensions. This gives you the flexibility to identify an element using one, two, or three subscripts (coordinates).
- In an array with two or three dimensions, the right-most dimension increments first in memory.

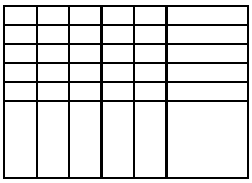
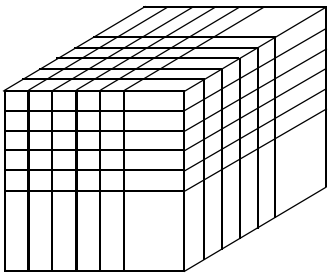


This array contains a two-dimensional grid of elements, six elements by six elements.

The right-most dimension increments to its maximum value then starts over.

When the right-most dimension starts over, the dimension to the left increments by one.

- The total number of elements in an array is the product of each dimension's size, as depicted in the following examples:

This array:	Stores data like:	For example:				
one dimension		Tag name:	Type	Dimension 0	Dimension 1	Dimension 2
		one_d_array	DINT[7]	7	--	--
		total number of elements = 7				
		valid subscript range DINT[x] where x=0–6				
two dimension		Tag name:	Type	Dimension 0	Dimension 1	Dimension 2
		two_d_array	DINT[4,5]	4	5	--
		total number of elements = 4 * 5 = 20				
		valid subscript range DINT[x,y] where x=0–3; y=0–4				
three dimension		Tag name:	Type	Dimension 0	Dimension 1	Dimension 2
		three_d_array	DINT[2,3,4]	2	3	4
		total number of elements = 2 * 3 * 4 = 24				
		valid subscript range DINT[x,y,z] where x=0–1; y=0–2, z=0–3				

- You can modify array dimensions when programming offline without loss of tag data. You cannot modify array dimensions when programming online.

application The combination of routines, programs, tasks, and I/O configuration used to define the operation of a single controller. See **project**.

B

base tag A tag that actually defines the memory where a data element is stored. See **alias tag**.

bidirectional connection A connection in which data flows in both directions: from the originator to the receiver and from the receiver to the originator. See **connection**, **unidirectional connection**.

binary Integer values displayed and entered in base 2 (each digit represents a single bit). Prefixed with 2#. Padded out to the length of the boolean or integer (1, 8, 16, or 32 bits). When displayed, every group of four digits is separated by an underscore for legibility. See **decimal**, **hexadecimal**, **octal**.

bit Binary digit. The smallest unit of memory. Represented by the digits 0 (cleared) and 1 (set).

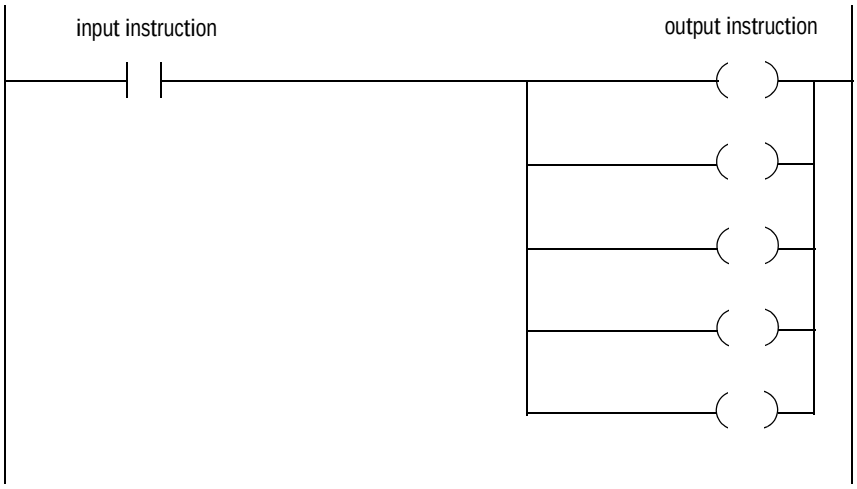
BOOL An data type that stores the state of a single bit, where:

- 0 equals off
- 1 equals on

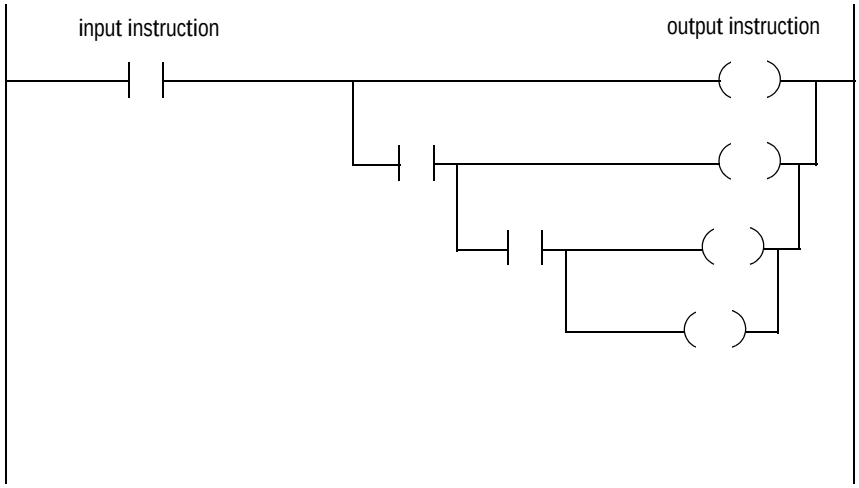
BOOL expression In Structured Text, an expression that produces either the BOOL value of 1 (true) or 0 (false).

- A bool expression uses bool tags, relational operators, and logical operators to compare values or check if conditions are true or false. For example, `tag1>65`.
- A simple bool expression can be a single BOOL tag.
- Typically, you use bool expressions to condition the execution of other logic.

branch There is no limit to the number of parallel branch levels that you can enter. The following figure shows a parallel branch with five levels. The main rung is the first branch level, followed by four additional branches.



You can nest branches to as many as 6 levels. The following figure shows a nested branch. The bottom output instruction is on a nested branch that is three levels deep.



byte A unit of memory consisting of 8 bits.

C

cache Depending on how you configure a MSG instruction, it may use a connection to send or receive data.

This type of message	And this communication method:	Uses a connection
CIP data table read or write	→	3
PLC2, PLC3, PLC5, or SLC (all types)	CIP	
	CIP with Source ID	
	DH+	3
CIP generic	→	your option (1)
block-transfer read or write	→	3

(1) You can connect CIP generic messages. But for most applications we recommend you leave CIP generic messages unconnected.

If a MSG instruction uses a connection, you have the option to leave the connection open (cache) or close the connection when the message is done transmitting.

If you:	Then:
Cache the connection	The connection stays open after the MSG instruction is done. This optimizes execution time. Opening a connection each time the message executes increases execution time.
Do not cache the connection	The connection closes after the MSG instruction is done. This frees up that connection for other uses.

If several messages go to the same device, the messages may be able to share a connection.

If the MSG instructions are to:	And they are:	Then:
different devices	→	Each MSG instruction uses 1 connection.
same device	enabled at the same time	Each MSG instruction uses 1 connection.
	NOT enabled at the same time	The MSG instructions share the connection. (I.e., Together they count as 1 connection.)

See **connection**, **uncached connection**.

change of state (COS) Any change in the status of a point or group of points on an I/O module.

CIP See **Control and Information Protocol**.

communication format Defines how an I/O module communicates with the controller. Choosing a communication format defines:

- what configuration tabs are available through the programming software
- the tag structure and configuration method

compatible module An electronic keying protection mode that requires that the vendor, catalog number, and major revision attributes of the physical module and the module configured in the software match in order to establish a connection to the module. See **disable keying**, **exact match**.

connection A communication link between two devices, such as between a controller and an I/O module, PanelView terminal, or another controller.

- Connections are allocations of resources that provide more reliable communications between devices than unconnected messages.
- The number of connections that a single controller can have is limited.
- You indirectly determine the number of connections the controller uses by configuring the controller to communicate with other devices in the system.

consumed tag A tag that receives the data that is broadcast by a produced tag over a ControlNet network or ControlLogix backplane. A consumed tag must be:

- controller scope
- same data type (including any array dimensions) as the remote tag (produced tag)

See ***produced tag***.

continuous task The task that runs continuously.

- The continuous task runs in the background. Any CPU time not allocated to other operations (such as motion, communications, and periodic tasks) is used to execute the programs within the continuous task.
- The continuous task restarts itself after the last of its programs finishes.
- A project does not require a continuous task.
- If used, there can be only one continuous task.
- All periodic tasks interrupt the continuous task.
- When you create a project, the default MainTask is the continuous task. You can leave this task as it is, or you can change its properties (name, type, etc.).

See ***periodic task***.

Control and Information Protocol Messaging protocol used by Allen-Bradley's Logix5000 line of control equipment. Native communications protocol used on the ControlNet network.

controller fault handler The controller fault handler is an optional task that executes when the:

- major fault is not an instruction-execution fault
- program fault routine:
  - could not clear the major fault
  - faulted
  - does not exist

You can create only one program for the controller fault handler. After you create that program, you must configure one routine as the main routine.

- The controller fault program does *not* execute a fault routine.
- If you specify a fault routine for the controller fault program, the controller never executes that routine.
- You can create additional routines and call them from the main routine.

controller scope Data accessible anywhere in the controller. The controller contains a collection of tags that can be referenced by the routines and alias tags in any program, as well as other aliases in the controller scope. See ***program scope***.

Coordinated System Time (CST) A 64-bit value that represents the number of microseconds since the CST master controller started counting.

- The CST value is stored as a DINT[2] array, where:
  - first element stores the lower 32 bits
  - second element stores the upper 32 bits
- You can use the CST timestamp to compare the relative time between data samples.

COUNTER Structure data type that contains status and control information for counter instructions

D

data type A definition of the memory size and layout that will be allocated when you create a tag of that data type.

decimal Integer values displayed and entered in base 10. No prefix. Not padded to the length of the integer. See **binary**, **hexadecimal**, **octal**.

description Optional text that you can use to further document your application.

- You can use any printable character, including carriage return, tab, and space.
- Descriptions do not download to the controller. They remain in the offline project file.
- Descriptions have these length limitations:
  - For tags, you can use up to 120 characters.
  - For other objects (tasks, programs, modules, etc.), you can use up to 128 characters.

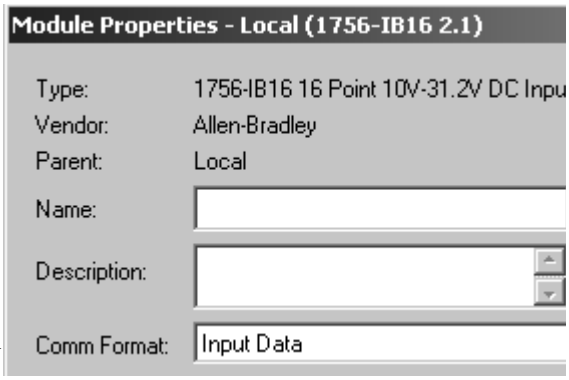
dimension Specification of the size of an array. Arrays can have as many as three dimensions. See ***array***.

DINT A data type that stores a 32-bit (4-byte) signed integer value (-2,147,483,648 to +2,147,483,647). In Logix5000 controllers, use DINTs for integers:

- Logix5000 controllers execute more efficiently and use less memory when working with 32-bit integers (DINTs) instead of 16-bit integers (INTs) or 8-bit integers (SINTs).
- Typically, instructions convert SINT or INT values to an **optimal data type** (usually a DINT or REAL value) during execution. Because this requires additional time and memory, minimize the use of the SINT and INT data types.

direct connection A direct connection is a real-time, data transfer link between the controller and an I/O module. The controller maintains and monitors the connection with the I/O module. Any break in the connection, such as a module fault or the removal of a module while under power, sets fault bits in the data area associated with the module.

A direct connection is any connection that *does not* use the Rack Optimization Comm Format.



See **rack-optimized connection**.

disable keying An electronic keying protection mode that requires no attributes of the physical module and the module configured in the software to match and still establishes a connection to the module. See **compatible module**, **exact match**.

download The process of transferring the contents of a project on the workstation into the controller. See **upload**.

E

elapsed time The total time required for the execution of all operations configured within a single task.

- If the controller is configured to run multiple tasks, elapsed time includes any time used/shared by other tasks performing other operations.
- While online, you can use the *Task Properties* dialog box to view the maximum scan time and the last scan time in ms for the current task. These values are elapsed time, which includes any time spent waiting for higher-priority tasks.

See **execution time**.

electronic keying A feature of the 1756 I/O line where modules can be requested to perform an electronic check to insure that the physical module is consistent with what was configured by the software. Enables the user via the software to prevent incorrect modules or incorrect revisions of modules from being inadvertently used. See **compatible module**, **disable keying**, **exact match**.

element An addressable unit of data that is a sub-unit of a larger unit of data. A single unit of an array.

- You specify an element in an array by its subscript(s):

For this array:	Specify:
one dimension	*array_name[subscript_0]*
two dimension	*array_name[subscript_0, subscript_1]*
three dimension	*array_name[subscript_0, subscript_1, subscript_2]*

See **array**.

exact match An electronic keying protection mode that requires that all attributes (vendor, catalog number, major revision, and minor revision) of the physical module and the module configured in the software match in order to establish a connection to the module.

execution time The total time required for the execution of a single program.

- Execution time includes only the time used by that single program, and excludes any time shared/used by programs in other tasks performing other operations.
- When online, use the Program Properties dialog box to view the maximum scan time and the last scan time (in ms) for the current program. These values are execution times for the program and do not include any time spent waiting for other programs or higher-priority tasks.

See ***elapsed time***.

exponential Real values displayed and entered in scientific or exponential format. The number is always displayed with one digit to the left of the decimal point, followed by the decimal portion, and then by an exponent. See ***style***.

F

faulted mode The controller generated a major fault, could not clear the fault, and has shut down. See ***major fault***.

float Real values displayed and entered in floating point format. The number of digits to the left of the decimal point varies according to the magnitude of the number. See ***style***.

H

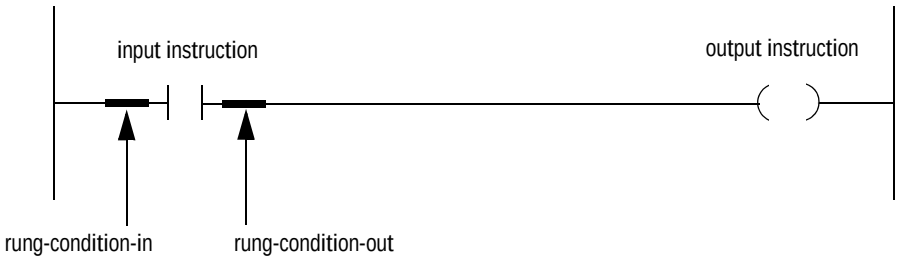
hexadecimal Integer values displayed and entered in base 16 (each digit represents four bits). Prefixed with 16#. Padded out to length of the boolean or integer (1, 8, 16, or 32 bits). When displayed, every group of four digits is separated by an underscore for legibility. See **binary**, **decimal**, **octal**.

I

immediate value An actual 32-bit signed real or integer value. Not a tag that stores a value.

index A reference used to specify an element within an array.

instruction The controller evaluates ladder instructions based on the rung condition preceding the instruction (rung-condition-in).



Only input instructions affect the rung-condition-in of subsequent instructions on the rung:

- If the rung-condition-in to an input instruction is true, the controller evaluates the instruction and sets the rung-condition-out to match the results of the evaluation.
  - If the instruction evaluates to true, the rung-condition-out is true.
  - If the instruction evaluates to false, the rung-condition-out is false.
- An output instruction does not change the rung-condition-out.
  - If the rung-condition-in to an output instruction is true, the rung-condition-out is set to true.
  - If the rung-condition-in to an output instruction is false, the rung-condition-out is set to false.

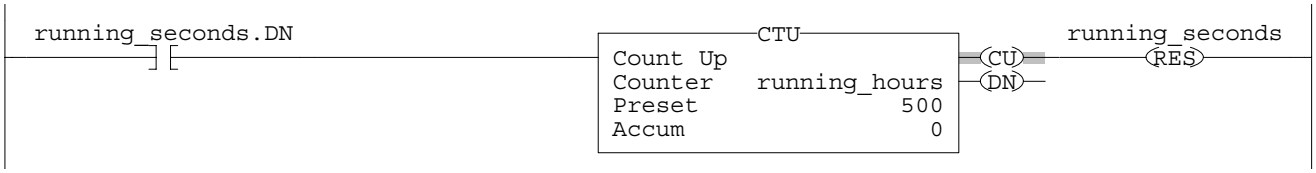
In Logix5000 controllers, you can enter multiple output instructions per rung of logic. You can enter the output instructions:

- in sequence on the rung (serial)
- between input instructions, as long as the last instruction on the rung is an output instruction

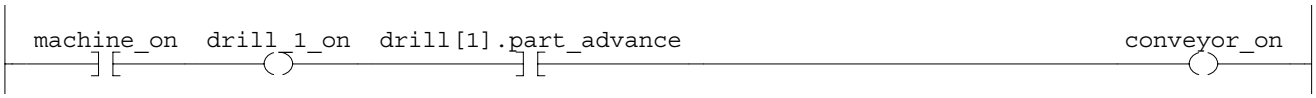
This example uses more than one output on a rung.

EXAMPLE Place multiple outputs on a rung

When *running_seconds.DN* turns on, *running_hours* counts up by one and *running_seconds* resets.



When *machine_on* turns on, turns on *drill_1_on*. When both *machine_on* and *drill[1].part_advance* are on, turns on *conveyor_on*.



42362

INT A data type that stores a 16-bit (2-byte) integer value (-32,768 to +32,767). Minimize your use of this data type:

- Typically, instructions convert SINT or INT values to an **optimal data type** (usually a DINT or REAL value) during execution. Because this requires additional time and memory, minimize the use of the SINT and INT data types.

interface module (IFM) A pre-wired I/O field wiring arm.

L

listen-only connection An I/O connection where another controller owns/provides the configuration data for the I/O module. A controller using a listen-only connection does not write configuration data and can only maintain a connection to the I/O module when the owner controller is actively controlling the I/O module. See **owner controller**.

load To copy a project from nonvolatile memory to the user memory (RAM) of the controller. This overwrites any project that is currently in the controller. See **nonvolatile memory**, **store**.

M

main routine The first routine to execute when a program executes. Use the main routine to call (execute) other routines (subroutines).

major fault A fault condition that is severe enough for the controller to shut down, unless the condition is cleared. When a major fault occurs, the controller:

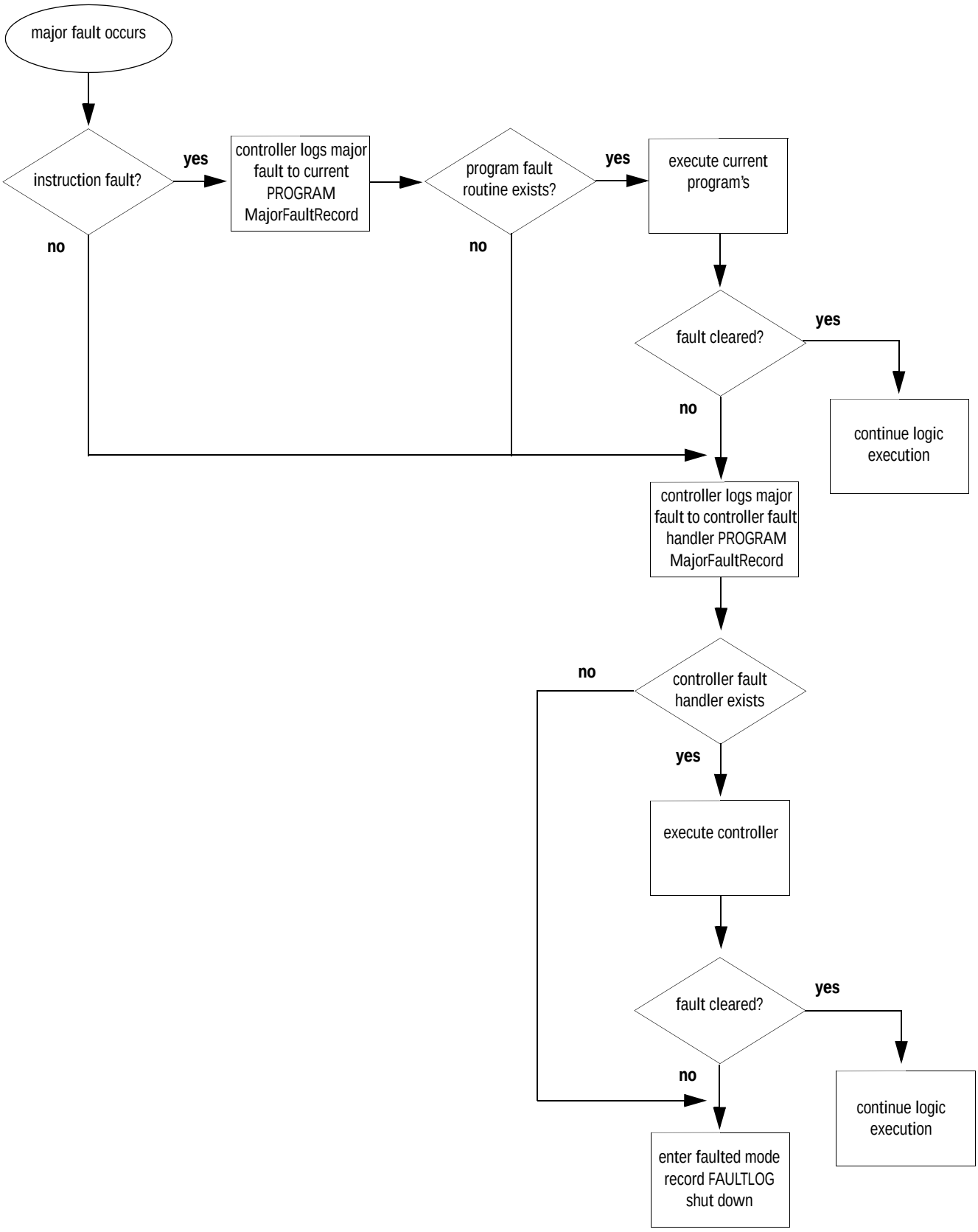
1. Sets a major fault bit
2. Runs user-supplied fault logic, if it exists
3. If the user-supplied fault logic cannot clear the fault, the controller goes to faulted mode
4. Sets outputs according to their output state during program mode
5. OK LED flashes red

The controller supports two levels for handling major faults:

- program fault routine:
  - Each program can have its own fault routine.
  - The controller executes the program's fault routine when an instruction fault occurs.
  - If the program's fault routine does not clear the fault or a program fault routine does not exist, the controller proceeds to execute the controller fault handler (if defined).
- controller fault handler:
  - If the controller fault handler does not exist or cannot clear the major fault, the controller enters faulted mode and shuts down. At this point, the FAULTLOG is updated. (See the next page.)
  - All non-instruction faults (I/O, task watchdog, etc.) execute the controller fault handler directly. (No program fault routine is called.)

The fault that was not cleared, and up to two additional faults that have not been cleared, are logged in the controller fault log.

See **faulted state**, **minor fault**.


major fault occurs
instruction fault?
yes
no
controller logs major fault to current PROGRAM MajorFaultRecord
program fault routine exists?
yes
no
execute current program's
fault cleared?
yes
no
continue logic execution
controller logs major fault to controller fault handler PROGRAM MajorFaultRecord
controller fault handler exists
no
yes
execute controller
fault cleared?
yes
no
continue logic execution
enter faulted mode record FAULTLOG shut down

major revision The 1756 line of modules have major and minor revision indicators. The major revision is updated any time there is a functional change to the module. See **electronic keying**, **minor revision**.

master (CST) Within a single chassis, one and only one, controller must be designated as the Coordinated System Time (CST) master. All other modules in the chassis synchronize their CST values to the CST master.

member An element of a structure that has its own data type and name.

- Members can be structures as well, creating nested structure data types.
- Each member within a structure can be a different data type.
- To reference a member in a structure, use this format:

  tag_name.member_name

  For example:

This address:	References the:
timer_1.pre	PRE value of the timer_1 structure.
input_load as data type load_info input_load.height	height member of the user-defined input_load structure

- If the structure is embedded in another structure, use the tag name of the structure at the highest level followed by a substructure tag name and member name:

  tag_name.substructure_name.member_name

  For example:

This address:	References the:
input_location as data type location input_location.load_info.height	height member of the load_info structure in the input_location structure.

- If the structure defines an array, use the array tag, followed by the position in the array and any substructure and member names.

  array_tag[position].member

  or

  array_tag[position].substructure_name.member_name

For example:

This address:	References the:
conveyor[10].source	*source* member of the 11th element in the *conveyor* array (array elements are zero based).
conveyor[10].info.height	height member of the info structure in the 11th element of the conveyor array (array elements are zero based).

See ***structure***.

memory Electronic storage media built into a controller, used to hold programs and data.

minor fault A fault condition that is *not* severe enough for the controller to shut down:

If this occurs:	The controller:
problem with an instruction	1. sets S:MINOR 2. logs information about the fault to the PROGRAM object, MinorFaultRecord attribute 3. sets bit 4 of the FAULTLOG object, MinorFaultBits attribute
periodic task overlap	sets bit 6 of the FAULTLOG object, MinorFaultBits attribute
problem with the serial port	sets bit 9 of the FAULTLOG object, MinorFaultBits attribute
low battery	sets bit 10 of the FAULTLOG object, MinorFaultBits attribute

To clear minor faults:

1. In the controller organizer, right-click the Controller *name_of_controller* folder and select Properties.
2. Click the Minor Faults tab.
3. Use the information in the Recent Faults list to correct the cause of the fault. Refer to "Minor Fault Codes" on page 17-418.
4. Click the Clear Minors button.

See ***major fault***.

minor revision The 1756 line of modules have major and minor revision indicators. The minor revision is updated any time there is a change to a module that does not affect its function or interface. See **electronic keying**, **major revision**.

multicast A mechanism where a module can send data on a network that is simultaneously received by more that one listener. Describes the feature of the ControlLogix I/O line which supports multiple controllers receiving input data from the same I/O module at the same time.

multiple owners A configuration setup where more than one controller has exactly the same configuration information to simultaneously own the same input module.

N

name Names identify controllers, tasks, programs, tags, modules, etc. Names follow IEC-1131-3 identifier rules and:

- must begin with an alphabetic character (A-Z or a-z) or an underscore (_)
- can contain only alphabetic characters, numeric characters, and underscores
- can have as many as 40 characters
- must not have consecutive or trailing underscore characters (_)
- are *not* case sensitive
- download to the controller

network update time (NUT) The repetitive time interval in which data can be sent on a ControlNet network. The network update time ranges from 2ms-100ms.

nonvolatile memory Memory of the controller that retains its contents while the controller is without power or a battery. See **load**, **store**.

numeric expression In Structured Text, an expression that calculates an integer or floating-point value.

- A numeric expression uses arithmetic operators, arithmetic functions, and bitwise operators. For example, `tag1+5`.
- Often, you nest a numeric expression within a bool expression. For example, `(tag1+5)>65`.

O

object A structure of data that stores status information. When you enter a GSV/SSV instruction, you specify the object and its attribute that you want to access. In some cases, there are more than one instance of the same type of object, so you might also have to specify the object name. For example, there can be several tasks in your application. Each task has its own TASK object that you access by the task name.

octal Integer values displayed and entered in base 8 (each digit represents three bits). Prefixed with 8#. Padded out to the length of the boolean or integer (1, 8, 16, or 32 bits). When displayed, every group of three digits is separated by an underscore for legibility. See **binary**, **decimal**, **hexadecimal**.

offline Viewing and editing a project that is on the hard disk of a workstation. See ***online***.

online Viewing and editing the project in a controller. See ***offline***.

optimal data type A data type that a Logix5000 instruction actually uses (typically the DINT and REAL data types).

- In the instruction set reference manuals, a **bold** data type indicates an optimal data type.
  - Instructions execute faster and require less memory if all the operands of the instruction use:
    - the same data type
    - an optimal data type
- If you mix data types and use tags that are not the optimal data type, the controller converts the data according to these rules
  - Are any of the operands a REAL value?

If:	Then input operands (e.g., source, tag in an expression, limit) convert to:
Yes	REALs
No	DINTs

  - After instruction execution, the result (a DINT or REAL value) converts to the destination data type, if necessary.
- Because the conversion of data takes additional time and memory, you can increase the efficiency of your programs by:
  - using the same data type throughout the instruction
  - minimizing the use of the SINT or INT data types

  In other words, use all DINT tags or all REAL tags, along with immediate values, in your instructions.

- This table summarizes how the controller converts data between data types:

<table>
<tr><th>Conversion:</th><th>Result:</th></tr>
<tr><td>larger integer to smaller integer</td><td>The controller truncates the upper portion of the larger integer and generates an overflow.

For example:
<table><tr><td></td><td>Decimal</td><td>Binary</td></tr><tr><td>DINT</td><td>65,665</td><td>0000_0000_0000_0001_0000_0000_1000_0001</td></tr><tr><td>INT</td><td>129</td><td>0000_0000_1000_0001</td></tr><tr><td>SINT</td><td>-127</td><td>1000_0001</td></tr></table></td></tr>
<tr><td>SINT or INT to REAL</td><td>No data precision is lost</td></tr>
<tr><td>DINT to REAL</td><td>Data precision could be lost. Both data types store data in 32 bits, but the REAL type uses some of its 32 bits to store the exponent value. If precision is lost, the controller takes it from the least-significant portion of the DINT.</td></tr>
<tr><td>REAL to integer</td><td>The controller rounds the fractional part and truncates the upper portion of the non-fractional part. If data is lost, the controller sets the overflow status flag.

Numbers round as follows:
• Numbers other than x.5 round to the nearest number.
• X.5 rounds to the nearest even number.

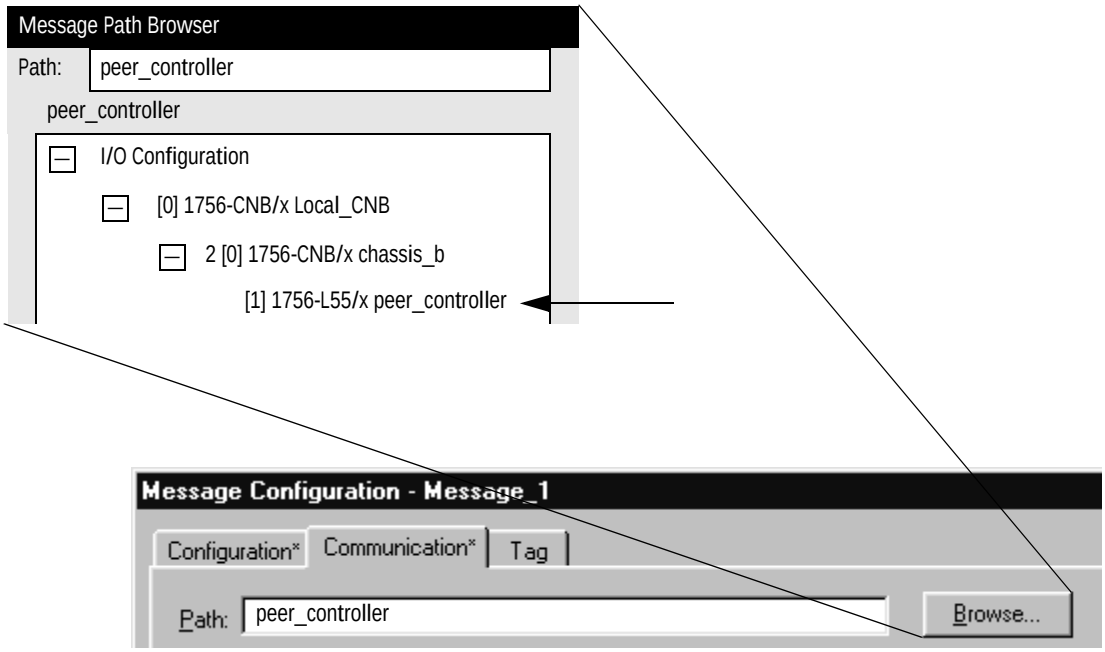
For example:
<table><tr><td>REAL (source)</td><td>DINT (result)</td></tr><tr><td>-2.5</td><td>-2</td></tr><tr><td>-1.6</td><td>-2</td></tr><tr><td>-1.5</td><td>-2</td></tr><tr><td>-1.4</td><td>-1</td></tr><tr><td>1.4</td><td>1</td></tr><tr><td>1.5</td><td>2</td></tr><tr><td>1.6</td><td>2</td></tr><tr><td>2.5</td><td>2</td></tr></table></td></tr>
</table>

overlap A condition where a task (periodic or event) is triggered while the task is still executing from the previous trigger.

owner controller The controller that creates the primary configuration and communication connection to a module. The owner controller writes configuration data and can establish a connection to the module. See **listen-only connection**.

P

path The path describes the route that a message takes to get to the destination. If the I/O configuration of the controller contains the destination device, use the Browse button to select the device. This automatically defines the path.



If the I/O configuration *does not* contain the destination device, then type the path to the destination using the following format:

port,address,port,address

Where:	For this:	Is:
port	backplane from any 1756 controller or module	1
	DF1 port from a Logix5000 controller	2
	ControlNet port from a 1756-CNB module	
	Ethernet port from a 1756-ENBx or -ENET module	
	DH+ port over channel A from a 1756-DHRIO module	
	DH+ port over channel B from a 1756-DHRIO module	3
address	ControlLogix backplane	slot number
	DF1 network	station address (0-254)
	ControlNet network	node number (1-99 decimal)
	DH+ network	8# followed by the node number (1-77 octal) For example, to specify the octal node address of 37, type 8#37.
	EtherNet/IP network	You can specify a module on an EtherNet/IP network using any of these formats: IP address (e.g., 130.130.130.5) IP address:Port (e.g., 130.130.130.5:24) DNS name (e.g., tanks) DNS name:Port (e.g., tanks:24)

See **connection**.

periodic task A task that is triggered by the operating system at a repetitive period of time.

- Use a periodic task for functions that require accurate or deterministic execution.
- Whenever the time expires, the task is triggered and its programs are executed.
- Data and outputs established by the programs in the task retain their values until the next execution of the task or they are manipulated by another task.
- You can configure the time period from 1 ms to 2000 s. The default is 10 ms.

ATTENTION



Ensure that the time period is longer than the sum of the execution times of all the programs assigned to the task. If the controller detects that a periodic task trigger occurs for a task that is already operating, a minor fault occurs.

- Periodic tasks always interrupt the continuous task.
- Depending on the priority level, a periodic task may interrupt other periodic tasks in the controller.

See **continuous task**.

periodic task overlap A condition that occurs when a task is executing and the same task is triggered again. The execution time of the task is greater than the periodic rate configured for the task. See ***periodic task***.

predefined structure A structure data type that stores related information for a specific instruction, such as the TIMER structure for timer instructions. Predefined structures are always available, regardless of the system hardware configuration. See **product defined structure**.

prescan Prescan is an intermediate scan during the transition to Run mode.

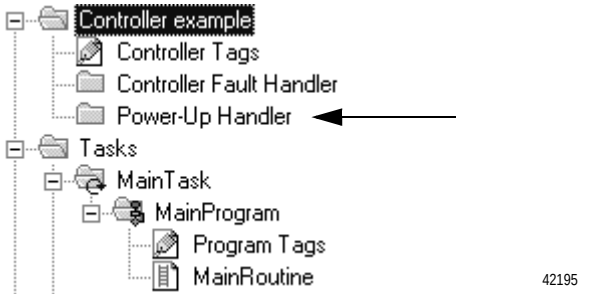
- The controller performs prescan when you change from Program mode to Run mode.
- The prescan examines all programs and instructions and initializes data based on the results.
- Some instructions execute differently during prescan than they do during the normal scan.

priority Specifies which task to execute first if two tasks are triggered at the same time.

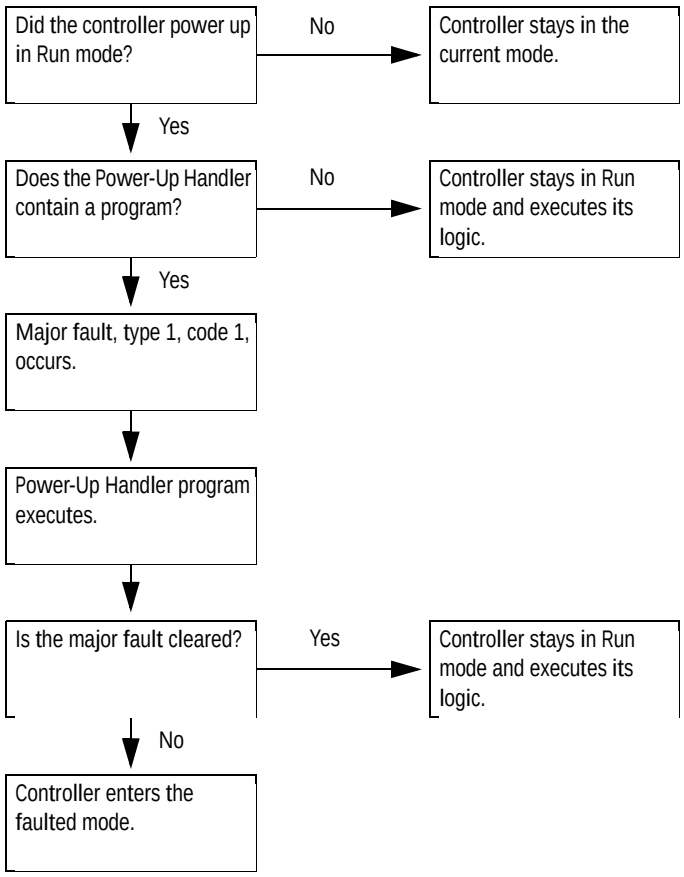
- The task with the higher priority executes first.
- Priorities range from 1-15, with 1 being the highest priority.
- A higher priority task will interrupt any lower priority task.
- If two tasks with the same priority are triggered at the same time, the controller switches between the tasks every millisecond.

postscan A function of the controller where the logic within a program is examined before disabling the program in order to reset instructions and data.

power-up handler An optional task that executes when the controller powers up in the Run mode. To use the Power-Up Handler, you must create a power-up program and associated main routine.



The Power-Up Handler executes as follows:



produced tag A tag that a controller is making available for use by other controllers. Produced tags are always at controller scope. See ***consumed tag***.

product defined structure A structure data type that is automatically defined by the software and controller. By configuring an I/O module you add the product defined structure for that module.

program A set of related routines and tags.

- Each program contains program tags, a main executable routine, other routines, and an optional fault routine.
- To execute the routines in a program, you assign (schedule) the program to a task:
  - When a task is triggered, the scheduled programs within the task execute to completion from first to last.
  - When a task executes a program, the main routine of the program executes first.
  - The main routine can, in turn, execute subroutines using the JSR instruction.
- The Unscheduled Programs folder contains programs that aren't assigned to a task.
- If the logic in the program produces a major fault, execution jumps to a configured fault routine for the program.
- The routines within a program can access the following tags:
  - program tags of the program
  - controller tags
- Routines cannot access the program tags of other programs.

See ***routine***, **task**.

program scope Data accessible only within the current program. Each program contains a collection of tags that can only be referenced by the routines and alias tags in that program. See***controller scope***.

project The file on your workstation (or server) that stores the logic, configuration, data, and documentation for a controller.

- The project file has an .ACD extension.
- When you create a project file, the file name is the name of the controller.
- The controller name is independent of the project file name. If you save a current project file as another name, the controller name is unchanged.
- If the name of the controller is different than the name of the project file, the title bar of the RSLogix 5000 software displays both names.

See **application**.

Q

qualifier In the action of a sequential function chart (SFC), a qualifier defines when an action starts and stops. See **action**, **Sequential Function Chart**, **step**.

R

rack-optimized connection For digital I/O modules, you can select rack-optimized communication. A rack-optimized connection consolidates connection usage between the controller and all the digital I/O modules in the chassis (or DIN rail). Rather than having individual, direct connections for each I/O module, there is one connection for the entire chassis (or DIN rail). See **direct connection**.

rate For a periodic task, the rate at which the controller executes the task, from 1 ms to 2,000,000 ms (2000 seconds). The default is 10 ms.

REAL A data type that stores a 32-bit (4-byte) IEEE floating-point value, with the following range:

- $-3.40282347E^{38}$ to $-1.17549435E^{-38}$ (negative values)
- 0
- $1.17549435E^{-38}$ to $3.40282347E^{38}$ (positive values)

The REAL data type also stores ±infinity, ±NAN, and -IND, but the software display differs based on the display format.

Display Format:	Equivalent:	
Real	+infinite	1.$
	- infinite	-1.$
	+NAN	1.#QNAN
	-NAN	-1.#QNAN
	-indefinite	-1.#IND
Exponential	+infinite	1.#INF000e+000
	- infinite	-1.#INF000e+000
	+NAN	1.#QNAN00e+000
	-NAN	-1.#QNAN00e+000
	-indefinite	-1.#IND0000e+000

The software also stores and displays the IEEE subnormal range:

- $-1.17549421E^{-38}$ to $-1.40129846E^{-45}$ (negative values)
- $1.40129846E^{-45}$ to $1.17549421E^{-38}$ (positive values)

removal and insertion under power (RIUP) A ControlLogix feature that lets a user to install or remove a module while chassis power is applied.

requested packet interval (RPI) When communicating over a the network, this is the maximum amount of time between subsequent production of input data.

- Typically, this interval is configured in microseconds.
- The actual production of data is constrained to the largest multiple of the network update time that is smaller than the selected RPI.
- Use a power of two times the ControlNet network update time (NUT).

  For example, if the NUT is 5 ms, type a rate of 5, 10, 20, 40 ms, etc.

See ***network update time (NUT)***.

routine A set of logic instructions in a single programming language, such as a ladder diagram.

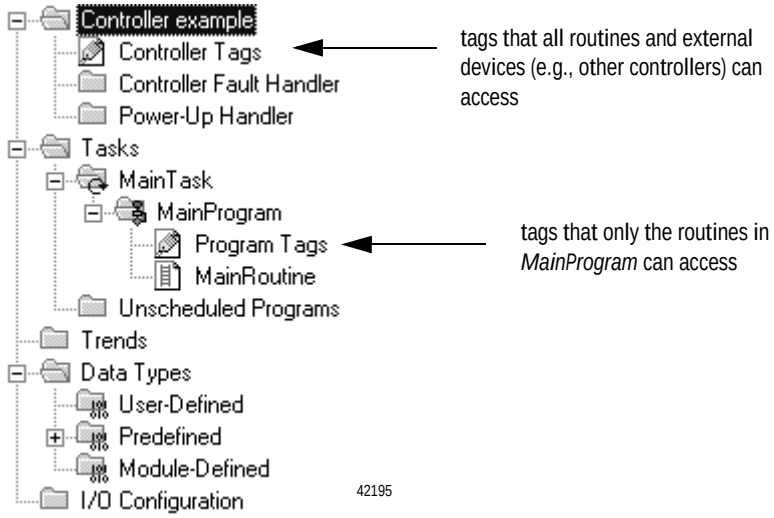
- Routines provide the executable code for the project in a controller (similar to a program file in a PLC or SLC controller).
- Each program has a main routine:
  - When the controller triggers the associated task and executes the associated program, the main routine is the first routine to execute.
  - To call another routine within the program, enter a JSR instruction in the main routine.
- You can also specify an optional program fault routine.
  - If any of the routines in the associated program produce a major fault, the controller executes program fault routine.

See ***program***, **task**.

S

scan time See ***elapsed time***, **execution time**.

scope Defines where you can access a particular set of tags. When you create a tag, you assign (scope) it as either a controller tag or a program tag for a specific program, as depicted below.



You can have multiple tags with the same name:

- Each tag must have a different scope. For example, one of the tags can be a controller tag and the other tags can be program tags for different programs. Or, each tag can be a program tag for a different program.
- Within a program, you cannot reference a controller tag if a tag of the same name exists as a program tag for that program.

See ***controller scope***, **program scope**.

Sequential Function Chart A sequential function chart (SFC) is similar to a flowchart. It uses steps and transitions to control a machine or process.

See **action**, **step**, **transition**.

SINT A data type that stores an 8-bit (1-byte) signed integer value (-128 to +127). Minimize your use of this data type:

- Typically, instructions convert SINT or INT values to an **optimal data type** (usually a DINT or REAL value) during execution. Because this requires additional time and memory, minimize the use of the SINT and INT data types.

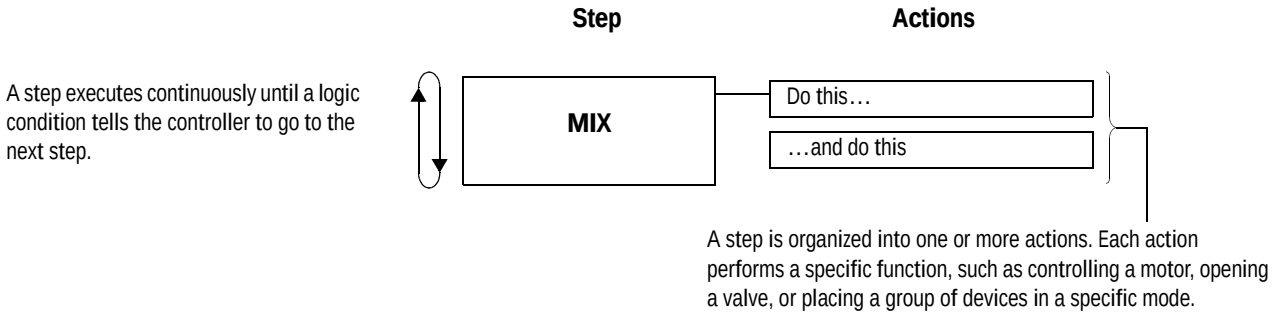
source key A mechanism that limits who can view a routine.

- You assign a source key to one or more routines.
- Source keys follow the same rules for names as other RSLogix 5000 components, such as routines, tags, and modules.
- To assign a source key to a routine (protect the routine), use RSLogix 5000 software. (You have to first activate the tool.).

- A source key file (sk.dat) stores the source keys. The source key file is separate from the RSLogix 5000 project files (.acd).
- To view a routine that is protected by a source key, you must have the source key.
- Without the source key, you cannot open a routine. RSLogix 5000 software displays "Source Not Available."
- Regardless of whether or not the source key is available, you can always download the project and execute all the routines.

See **name**.

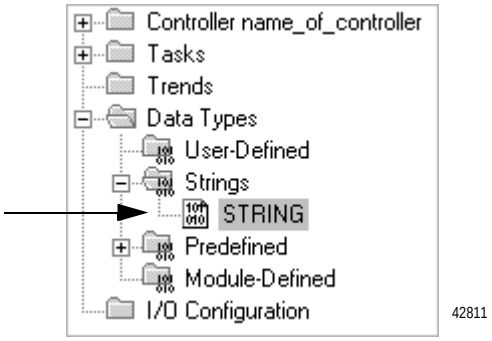
step In a sequential function chart (SFC), a step represents a major function of a process. It contains the events that occur at a particular time, phase, or station.



See **action**, **Sequential Function Chart**, **transition**.

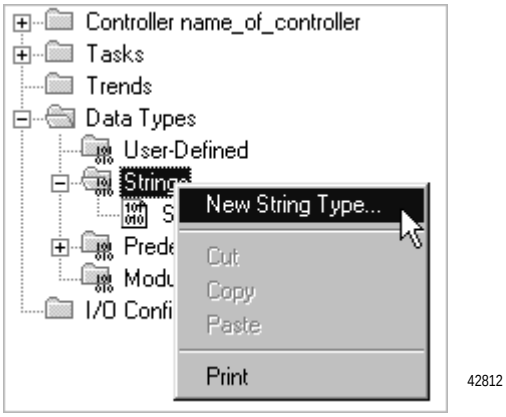
store To copy a project to the nonvolatile memory of the controller. This overwrites any project that is currently in the nonvolatile memory. See **load, nonvolatile memory**.

string A group of data types that store ASCII characters.



You can use the default STRING data type. It stores up to 82 characters.

or



You can create a new string data type to store the number of characters that you define.

Each string data type contains the following members:

Name:	Data Type:	Description:	Notes:
LEN	DINT	number of characters in the string	The LEN automatically updates to the new count of characters whenever you: • use the String Browser dialog box to enter characters • use instructions that read, convert, or manipulate a string The LEN shows the length of the current string. The DATA member may contain additional, old characters, which are not included in the LEN count.
DATA	SINT array	ASCII characters of the string	• To access the characters of the string, address the name of the tag. For example, to access the characters of the string_1 tag, enter string_1. • Each element of the DATA array contains one character. • You can create new string data types that store less or more characters.

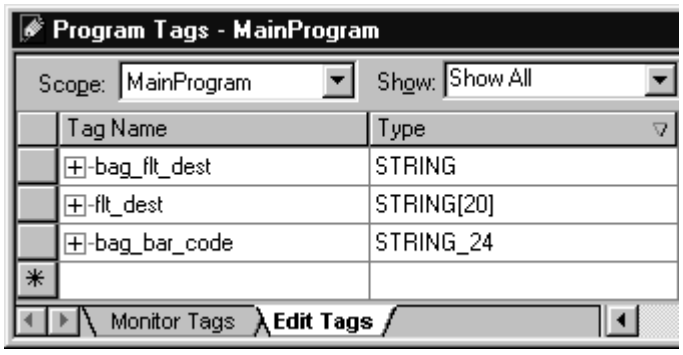
New string data types are useful in the following situations:

- If you have a large number of strings with a fixed size that is less than 82 characters, you can conserve memory by creating a new string data type.
- If you must handle strings that have more than 82 characters, you can create a new string data type to fit the required number of characters.

IMPORTANT Use caution when you create a new string data type. If you later decide to change the size of the string data type, you may lose data in any tags that currently use that data type.

If you:	Then:
make a string data type smaller	• The data is truncated. • The LEN is unchanged.
make a string data type larger	The data and LEN is reset to zero.

This example shows the STRING data type and a new string data type.



This tag uses the default STRING data type.

This tag is an 20 element array of the default STRING data type.

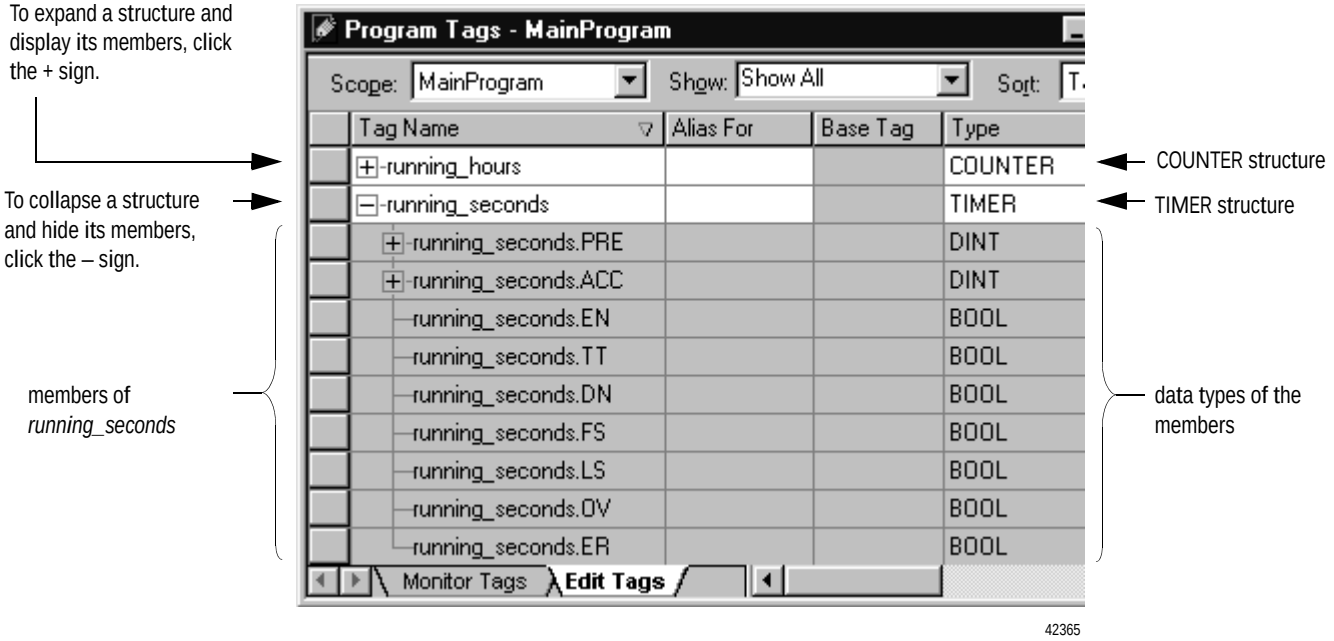
This tag uses a new string data type.

- The user named the string data type STRING_24.
- The new string data type stores only 24 characters.

structure Some data types are a structure.

- A structure stores a group of data, each of which can be a different data type.
- Within a structure, each individual data type is called a **member**.
- Like tags, members have a name and data type.
- You create your own structures, called a **user-defined data type**, using any combination of individual tags and most other structures.
- To copy data to a structure, use the COP instruction. See the *Logix5000 Controllers General Instruction Set Reference Manual*, publication 1756-RM003.

The COUNTER and TIMER data types are examples of commonly used structures.



See ***member***, **user-defined data type**.

style The format that numeric values are displayed in. See ***ASCII***, **binary**, **decimal**, **exponential**, **float**, **hexadecimal**, **octal**.

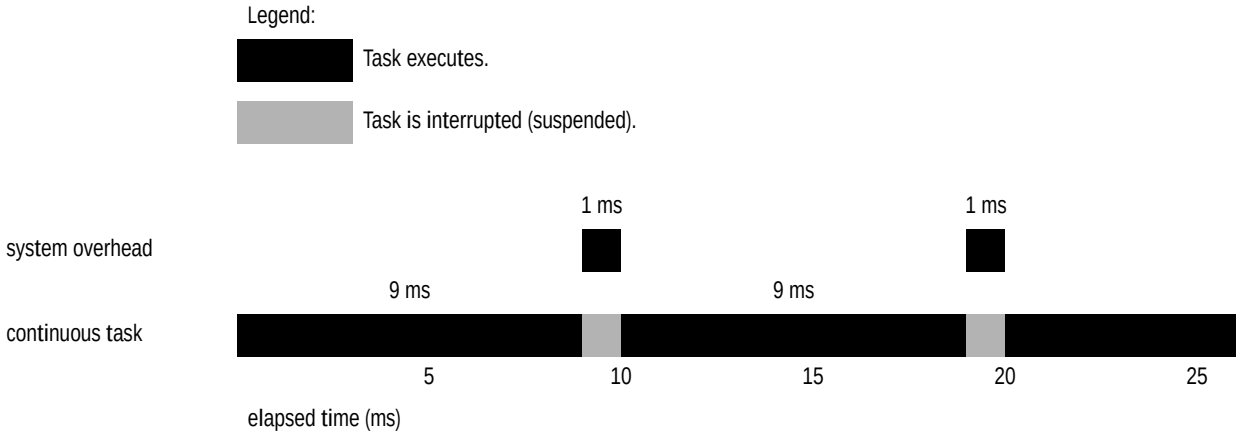
system overhead time slice Specifies the percentage of controller time (excluding the time for periodic tasks) that is devoted to communication and background functions (system overhead):

- The controller performs system overhead functions for up to 1 ms at a time.
- If the controller completes the overhead functions in less than 1 ms, it resumes the continuous task.
- Communication and background functions include the following:
  - communicate with programming and HMI devices (such as RSLogix 5000 software)
  - respond to messages
  - send messages, including block-transfers
  - re-establish and monitor I/O connections (such as RIUP conditions); this *does not* include normal I/O communications that occur during program execution
  - bridge communications from the serial port of the controller to other ControlLogix devices via the ControlLogix backplane
- If communications are not completing fast enough, increase the system overhead timeslice.

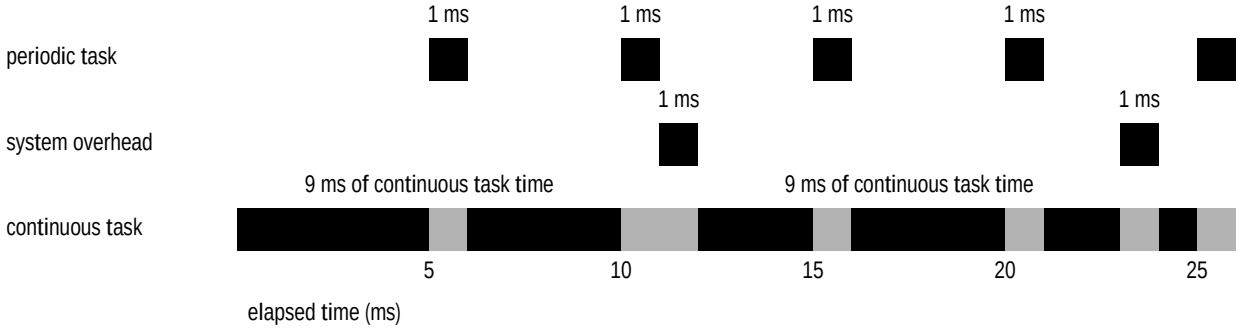
This table shows the ratio between the continuos task and the system overhead functions:

At this time slice:	The continuous tasks runs for:	And then overhead occurs for up to:
10%	9 ms	1 ms
20%	4 ms	1 ms
33%	2 ms	1 ms
50%	1 ms	1 ms

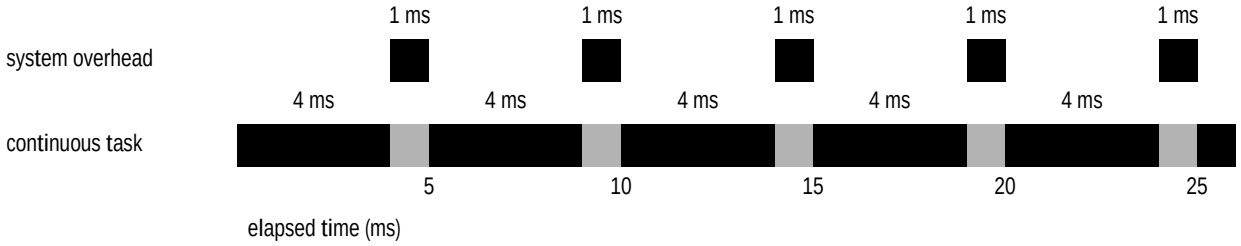
At the default time slice of 10 %, system overhead interrupts the continuous task every 9 ms (of continuous task time).



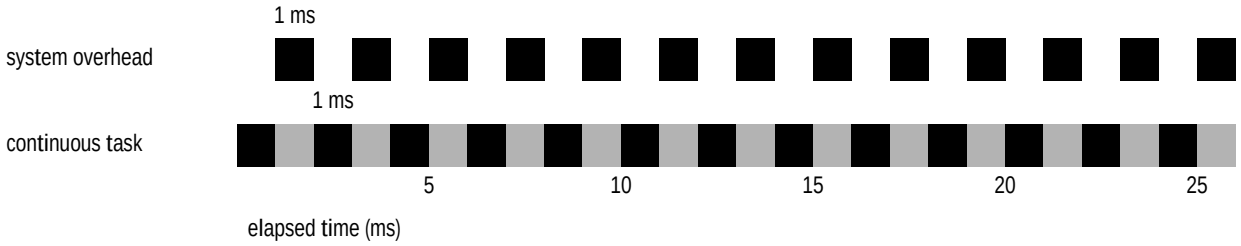
The interruption of a periodic task increases the elapsed time (clock time) between the execution of system overhead.



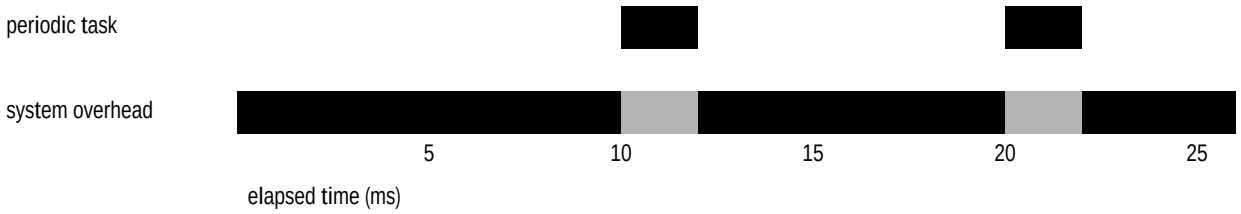
If you increase the time slice to 20 %, the system overhead interrupts the continuous task every 4 ms (of continuous task time).



If you increase the time slice to 50 %, the system overhead interrupts the continuous task every 1 ms (of continuous task time).



If the controller only contains a periodic task (s), the system overhead timeslice value has no effect. System overhead runs whenever a periodic task is not running.



To change the system overhead time slice:

1. Open the RSLogix 5000 project.
2. In the controller organizer, right-click the Controller *name_of_controller* folder and select Properties.
3. Click the Advanced tab.
4. In the System Overhead Time Slice text box, type or select the percentage of overhead time (10 -90%).
5. Click *OK*.

T

tag A named area of the controller's memory where data is stored.

- Tags are the basic mechanism for allocating memory, referencing data from logic, and monitoring data.
- The minimum memory allocation for a tag is four bytes.
  - When you create a tag that stores a BOOL, SINT, or INT (which are smaller than four bytes), the controller allocates four bytes, but the data only fills the part it needs.
  - User-defined data types and arrays store data in contiguous memory and pack smaller data types into 32-bit words.

The following examples show memory allocation for various tags:

– start, which uses the BOOL data type:

Memory allocation	Bits	
	31 ... 1	0
allocation	not used	start

– station_status, which uses the DINT data type:

Memory allocation:	Bits	
	31	0
allocation	station_status	

– mixer, which uses a user-defined data type:

Memory allocation	Bits			
	31 ... 24	23 ... 16	15 ... 8	7 ... 0
allocation 1	mixer.pressure			
allocation 2	mixer.temp			
allocation 3	mixer.agitate_time			
allocation 4	unused	unused	unused	bit 0 mixer.inlet bit 1 mixer.drain bit 2 mixer.agitate

– temp_buffer, which is an array of four INTS (INT[4]):

Memory allocation:	Bits	
	31 ... 16	0
allocation 1	*temp_buffer[1]*	temp_buffer[0]
allocation 2	temp_buffer[3]	temp_buffer[2]

See ***alias tag***, **base tag**, **consumed tag**.

task A scheduling mechanism for executing a program.

- By default, each new project file contains a pre-configured continuous task.
- You configure additional, periodic tasks, as needed.
- A task provides scheduling and priority information for a set of one or more programs that execute based on specific criteria.
- Once a task is triggered (activated), all the programs assigned (scheduled) to the task execute in the order in which they are displayed in the controller organizer.
- You can only assign a program to one task at a time.

See ***continuous task***, **periodic task**.

timestamp A ControlLogix process that records a change in input data with a relative time reference of when that change occurred.

transition In a sequential function chart (SFC), a transition is the true or false condition or conditions that determine when to go to the next step.

U

uncached connection With the MSG instruction, an uncached connection instructs the controller to close the connection upon completion of the mSG instruction. Clearing the connection leaves it available for other controller uses. See **connection**, **cached connection**.

unidirectional connection A connection in which data flows in only one direction: from the originator to the receiver. See **connection**, **bidirectional connection**.

upload The process of transferring the contents of the controller into a project file on the workstation.

If you do not have the project file for a controller, you can upload from the controller and create a project file. However, not everything that is stored in a project file is available from the controller. If you upload from a controller, the new project file will not contain:

- rung comments
- descriptions for tags, tasks, programs, routines, modules, or user-defined structures
- chains of aliases (aliases pointing to other aliases)

  Alias chains are not completely reconstructed from the controller. If there are several possible names for a data item, the firmware and software choose a best-fit alias that may not reflect how the alias was specified in the original project.

See **download**.

user-defined data type You can also create your own **structures**, called a user-defined data type (also commonly referred to as a user-defined structure). A user-defined data type groups different types of data into a single named entity.

- Within a user-defined data type, you define the **members**.
- Like tags, members have a name and data type.
- You can include arrays and structures.
- Once you create a user-defined data type, you can create one or more tags using that data type.
- Minimize your use of the following data type because they typically increase the memory requirements and execution time of your logic:
  - **INT**
  - **SINT**

  For example, some system values use the SINT or INT data type. If you create a user-defined data type to store those values, then use the corresponding SINT or INT data type.

- If you include members that represent I/O devices, you must use ladder logic to copy the data between the members in the structure and the corresponding I/O tags. See "Buffering I/O" on page 1-28.
- When you use the BOOL, SINT, or INT data types, place members that use the same data type in sequence:

more efficient

BOOL
BOOL
BOOL
DINT
DINT

less efficient

BOOL
DINT
BOOL
DINT
BOOL

- You can use single dimension arrays.
- You can create, edit, and delete user-defined data types only when programming offline.
- If you modify a user-defined data type and change its size, the existing values of any tags that use the data type are set to zero (0).
- To copy data to a structure, use the COP instruction. See the *Logix5000 Controllers General Instruction Set Reference Manual*, publication 1756-RM003.

See ***structure***.

W

watchdog Specifies how long a task can run before triggering a major fault.

- Each task has a watchdog timer that monitors the execution of the task.
- A watchdog time can range from 1 ms to 2,000,000 ms (2000 seconds). The default is 500 ms.
- The watchdog timer begins to time when the task is initiated and stops when all the programs within the task have executed.
- If the task takes longer than the watchdog time, a major fault occurs: (The time includes interruptions by other tasks.)
- A watchdog time-out fault (major fault) also occurs if a task triggered again while it is executing (periodic task overlap). This can happen if a lower-priority task is interrupted by a higher-priority task, delaying completion of the lower-priority task.
- You can use the controller fault handler to clear a watchdog fault. If the same watchdog fault occurs a second time during the same logic scan, the controller enters faulted mode, regardless of whether the controller fault handler clears the watchdog fault.

ATTENTION



If the watchdog timer reaches a configurable preset, a major fault occurs. Depending on the controller fault handler, the controller might shut down.

To change the watchdog time of a task:

1. Open the RSLogix 5000 project.
2. In the controller organizer, right-click **name_of_task** and select Properties.
3. Click the Configuration tab.
4. In the Watchdog text box, type a watchdog time.
5. Click OK.

How Are We Doing?

Your comments on our technical publications will help us serve you better in the future. Thank you for taking the time to provide us feedback.

You can complete this form and mail it back to us, visit us online at www.ab.com/manuals, or email us at RADocumentComments@ra.rockwell.com

vr

Pub. Title/Type Logix5000 Controllers Common Procedures

Cat. No. 1756 ControlLogix, 1769 CompactLogix, 1789 SoftLogix, 1794 FlexLogix, PowerFlex 700S with DriveLogix

Pub. No. 1756-PM001I-EN-P Pub. Date January 2007 Part No. 953030-27

Please complete the sections below. Where applicable, rank the feature (1=needs improvement, 2=satisfactory, and 3=outstanding).

Overall Usefulness	1	2	3	How can we make this publication more useful for you?
Completeness (all necessary information is provided)	1	2	3	Can we add more information to help you? procedure/step illustration feature example guideline other explanation definition
Technical Accuracy (all provided information is correct)	1	2	3	Can we be more accurate? text illustration
Clarity (all provided information is easy to understand)	1	2	3	How can we make things clearer?
Other Comments				You can add additional comments on the back of this form.

Your Name _____ Location/Phone _____

Your Title/Function _____

Would you like us to contact you regarding your comments?

___No, there is no need to contact me

___Yes, please call me

___Yes, please email me at _____

___Yes, please contact me via _____

Return this form to: Allen-Bradley Marketing Communications, 1 Allen-Bradley Dr., Mayfield Hts., OH 44124-9705

Phone: 440-646-3176 Fax: 440-646-3525 Email: RADocumentComments@ra.rockwell.com

PLEASE FASTEN HERE (DO NOT STAPLE)

Other Comments

PLEASE FOLD HERE

PLEASE REMOVE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST-CLASS MAIL PERMIT NO. 18235 CLEVELAND OH

POSTAGE WILL BE PAID BY THE ADDRESSEE



Rockwell Automation

1 ALLEN-BRADLEY DR
MAYFIELD HEIGHTS OH 44124-9705

Numerics

1784-CF64 Industrial CompactFlash card
format 20-447
stroage of firmware 20-448
use of CompactFlash reader 20-458

A

action 5-182
add 5-180
assign order 5-184
assign qualifier 5-180
boolean 4-137
choose between boolean and non-boolean 4-135
configure 5-180
data type 4-138
non-boolean 4-136
program 4-136, 5-182
qualifier 4-139
reset 4-155
store 4-155
use expression 5-181
use of structured text 5-182
address
assign indirect 2-54
tag 2-50
function block diagram 8-233
I/O module 1-27
ladder logic 7-224, 7-227
alarm
sequential function chart 4-145, 5-177
alias
create 2-53
show/hide 2-52
use of 2-51
arithmetic operators
structured text 6-196
array
calculate subscript 2-56
create 2-43
index through 2-54
organize 2-36
overview 2-39
ASCII
build string 14-385
compare characters 14-379, 14-382
configure serial port 13-367
configure user protocol 13-368
connect device 13-366
convert characters 14-383
decode message 14-384
enter characters 13-375
extract characters 14-378
look up characters 14-379
manipulate characters 14-377
organize data 13-370
read characters 13-371
structured text assignment 6-194
write characters 13-373
assignment
ASCII character 6-194
non-retentive 6-193
retentive 6-192
assume data available 8-238, 8-240, 8-247
automatic reset
sequential function chart 4-153

B

bar code
extract characters 14-378
search for a match 14-379
test characters 14-379, 14-382
bitwise operators
structured text 6-200
block transfer
guidelines 11-352
block. See array
BOOL expression
sequential function chart 4-143, 5-178
structured text 6-194
boolean action 4-137, 5-182
program 4-137
branch
ladder logic 7-220
sequential function chart 4-131, 5-171, 5-173
buffer
for unconnected message 11-351, 11-353
I/O 1-28

C

cache
connection 11-350
CASE 6-206
change of state
configure for I/O module 3-79
clear
major fault 16-399
nonvolatile memory 20-456
codes
major fault 16-412

minor fault 17-418
comments
structured text 6-218
communicate
other controllers 11-331
with multiple controllers B-473
communication
execution 3-115
guidelines for unscheduled communication 3-65
I/O module 1-22
impact on execution 3-63
Message instruction 11-347
system overhead time slice 3-115
CompactFlach card
use of reader 20-458
compare
ASCII characters 14-379, 14-382
compliance tables C-487
configure
action 5-180
alarm 5-177
electronic keying 1-26
execution of sequential function chart 4-162, 5-188
I/O module 1-21, 11-332
load from nonvolatile memory 20-449, 20-453
output processing for a task 3-70
serial port for ASCII 13-367
step 5-176
system overhead time slice 3-115
user protocol for ASCII 13-368
connection
cache 11-350
direct 1-23
failure 11-334
I/O fault 11-334
inhibit 11-332
listen-only 1-24
monitor 11-336
overview 1-22
produced or consumed tag 11-340
rack-optimized 1-23
reduce the number of 1-23
construct
structured text 6-202
consume
tag 11-339
consumed tag
connection requirements 11-340
create 11-344
maintain data integrity 3-100
organize 11-341
overview 11-339
synchronize controllers 3-101
continuous task
overview 3-60
use of 3-60
controller
memory information 21-463
nonvolatile memory 20-445, 20-446
number of tasks 3-61
shut down 16-409
suspend 16-409
synchronize 3-101
tags 2-35
triggers supported 3-78
update firmware
during load from nonvolatile memory 20-448
controller tags
use of 2-35
ControlNet
bandwidth limits 11-342
produce and consume data 11-339
convert
ASCII characters 14-383
COS. See change of state
create
alias 2-53
consumed tag 11-344
event task 3-109
periodic task 3-110
produced tag 11-343
string 14-385
string data type 13-370
tag 2-39, 7-227
function block diagram 8-248
user-defined data type 2-46

D

data
ASCII 13-370
block. See array
definitions C-484
enter ASCII characters 13-375
force 15-391, 15-392
I/O 1-27
produce and consume 11-339
data table. See tag
data type
choose 2-33
convert data 11-356
overview 2-33

structure 2-33
data. See also tag
description
structured text 6-218
tag 2-48
user-defined data type 2-48
disable
force 15-389, 15-396
document
sequential function chart 5-184
structured text 6-218
tag 2-48
user-defined data type 2-48
documentation
show or hide in sequential function chart 5-187
don"t scan
sequential function chart 4-150

E

electronic keying 1-26
enable
force 15-388
enter
action 5-180
address 7-227
ASCII characters 13-375
function block element 8-245
ICON 8-251
ladder logic 7-225
OCON 8-250
selection branch 5-173
sequential function chart 5-169
simultaneous branch 5-171
EOT instruction 4-144
Ethernet
produce and consume tags 11-339
event task
axis registration trigger 3-90
axis watch trigger 3-94
checklist for consumed tag event 3-102, 3-103
checklist for input event 3-83
checklist for motion group event 3-89
checklist for registration event 3-91
checklist for watch position event 3-95
choose trigger 3-77
consumed tag trigger 3-98
create 3-109
estimate throughput 3-85
EVENT trigger 3-106
input data trigger 3-79
motion group trigger 3-88
overview 3-60
timeout 3-111
use of 3-60
execute
event task 3-77
execution
sequential function chart 4-163, 5-188
execution order
function block diagram 8-235
expression
BOOL expression
sequential function chart 4-143, 5-178
structured text 6-194
calculate array subscript 2-56
numeric expression
sequential function chart 5-177, 5-181
structured text 6-194
order of execution
structured text 6-200
structured text
arithmetic operators 6-196
bitwise operators 6-200
functions 6-196
logical operators 6-199
overview 6-194
relational operators 6-197
extract
ASCII characters 14-378

F

fault
clear 16-399
communication loss 11-334
create user-defined 16-409
develop routine to clear fault 16-399
during load from nonvolatile memory 20-447
during prescan 16-405
I/O connection 11-334
indirect address 16-405
major fault codes 16-412
minor fault codes 17-418
monitor minor 17-415
test a fault routine 16-408
feedback loop
function block diagram 8-238
file. See array
firmware

update during load from nonvolatile memory 20-448
first scan bit 18-421
FOR…DO 6-209
force
disable 15-389, 15-396
enable 15-388
LED 15-389
monitor 15-389
options 15-391
remove 15-389, 15-396
safety precautions 15-388
sequential function chart 15-393, 15-395
tag 15-391, 15-392
function block diagram
add an element 8-245
add sheet 8-245
assign immediate value 8-249
choose elements 8-232
connect elements 8-247
create a scan delay 8-240
force a value 15-387
hide a pin 8-246
latching data 8-235
order of execution 8-235
organize sheets 8-231
resolve a loop 8-238
resolve data flow between blocks 8-240
resolve loop 8-247
show a pin 8-246
functions
structured text 6-196

G

global data. See scope

I

I/O
buffer 1-28
document. See alias
impact on execution 3-63
output processing 3-70
synchronize with logic 1-28
throughput for event task 3-85
update period 1-22
I/O module
choose for event task 3-82
communication format 1-23
communication loss 11-334
configure 1-21
configure change of state 3-79
connection fault 11-334
electronic keying 1-26
inhibit 11-332
ownership 1-24
tag address 1-27
trigger event task 3-79
update period 1-22
ICON
add 8-251
choosing 8-232
enter 8-245
IEC61131-3 compliance
data definitions C-484
instruction set C-486
introduction C-483
operating system C-484
program portability C-486
programming language C-485
tables C-487
IF...THEN 6-203
immediate value
function block diagram 8-249
ladder logic 7-228
index. See indirect address
indirect address 2-54
clear a major fault 16-405
format 2-50
use of expression 2-56
inhibit
connetion 11-332
I/O module 11-332
task 3-74
instruction set C-486
IREF
choosing 8-232
enter 8-245
latching data 8-235
to assign immediate value 8-249

J

jump
sequential function chart 4-134

K

keying
electronic 1-26

L

ladder logic

arrange input instructions 7-223
arrange output instructions 7-224
assign immediate value 7-228
branch 7-220
develop 7-222
enter 7-225
force a value 15-387
manage messages A-469
override a value 15-387
rung condition 7-221
last scan
sequential function chart 4-148
latching data
function block diagram 8-235
LED
force 15-389
load a project 20-453
local data. See scope
logical operators
structured text 6-199
look up a bar code 14-379

M

main routine
use of sequential function chart 4-126
major fault
codes 16-412
create user-defined 16-409
develop fault routine 16-399
manipulate string 14-377
mark as boolean 5-182
math operators
structured text 6-196
memory
allocation for tags 2-33
determine amount of free 21-463
types 21-463
message
cache connecton 11-350
convert between 16 and 32-bit data 11-356
decode string 14-384
guidelines 11-352
limits 11-349
manage multiple messages A-469
processing 11-348
queue 11-349
to a single controller 11-347
to multiple controllers B-473
unconnected buffer 11-351, 11-353
minor fault
codes 17-418
logic 17-415
monitor
forces 15-389
I/O connection 11-336
task 3-67
motion planner
impact on execution 3-63
trigger event task 3-88

N

name
guidelines for tag 2-36
reuse of tag name 2-35
tag name 7-224, 8-233
nonvolatile memory
check for a load 20-455
clear 20-456
fault during load 20-447
load a project 20-453
load image options 20-449
overview 20-445
store a project 20-451
supported controllers 20-446
numeric expression 5-177, 5-181, 6-194

O

OCON
add 8-250
choosing 8-232
enter 8-245
operating system C-484
operators
order of execution
structured text 6-200
order of execution
function block diagram 8-235
structured text expression 6-200
OREF
choosing 8-232
enter 8-245
organize
strings 13-370
output processing
manually configure 3-72
overview 3-70
programmatically configure 3-73
overlap
manually check for 3-67
overview 3-66
programmaticlly check for 3-68

overrun. See overlap
ownership
I/O module 1-24

P

pass-through description 2-48
pause an SFC 4-163
periodic task
application for 4-125
create 3-110
overview 3-60
use of 3-60
PLC-5C
share data 11-346
postscan
sequential function chart 4-148
structured text 6-193
prescan
clear a major fault 16-405
prgoram tags
use of 2-35
priority
assign 3-62
selection branch 5-174
produce
tag 11-339
produced tag
connection requirements 11-340
create 11-343
organize 11-341
overview 11-339
program
action 4-136, 5-182
boolean action 4-137
portability C-486
tags 2-35
transition 5-178
program/operator control
overview 8-242
programmatic reset option 4-151
programming language
IEC61131-3 compliance C-485
project
load from nonvolatile memory 20-449, 20-453
nonvolatile memory 20-445
number of tasks 3-61
organize tasks 3-60
protect 19-425, 19-442
restrict access 19-442
store in nonvolatile memory 20-451
protect
project 19-425, 19-442
routine 19-431

Q

qualifier
assign 5-180
choose 4-139

R

read
ASCII characters 13-371
registration
trigger event task 3-90
relational operators
structured text 6-197
remove
force 15-389, 15-396
REPEAT...UNTIL 6-215
requested packet interval 1-22
reset
action 4-155
SFC 4-159
reset an SFC 4-161, 4-163
restart
sequential function chart 4-159
routine
as transition 4-144
nest within sequential function chart 4-161
protect 19-431
restrict access 19-431
verify 5-189, 7-229, 8-251
routine source protection 19-431
RPI. See requested packet interval
RSI Security Server software 19-442
RSLogix 5000 Source Protection tool 19-431
rung condition 7-221

S

scan delay
function block diagram 8-240
scope
guidelines 2-36
tag 2-35
security
protect a project 19-442
protect a routine 19-431

Security Server software 19-442
selection branch
assign priorities 5-174
create 5-173
overview 4-133
send
ASCII characters 13-373
sequential function chart
action
assign order 5-184
call a subroutine 5-183
configure 5-180
enter 5-180
overview 4-135
program 5-182
use of boolean action 4-137
automatic reset option 4-153
boolean action 4-137
call a subroutine 5-183
configure execution 5-188
define tasks 4-125
document 5-184
don"t scan option 4-150
enter a new element 5-169
execution
configure 4-162
diagrams 4-163
pause 4-163
force element 15-387, 15-393, 15-395
last scan 4-148
nest 4-161
numeric expression 5-177, 5-181
organize a project 4-126
organize steps 4-131
pause an SFC 4-163
programmatic reset option 4-151
qualifier 4-139
reset
data 4-148
SFC 4-159, 4-161, 4-163
restart 4-159
return to previous step 5-175
selection branch
assign priorities 5-174
create 5-173
overview 4-133
sequence 4-133
show or hide documentation 5-187
simultaneous branch
create 5-171
overview 4-134
step
configure 5-176
define 4-126
organize 4-131
overview 4-126
step through
simultaneous branch 15-393
transition 15-393
step through simultaneous branch 15-393
step through transition 15-393
stop 4-158
text box 5-186
transition
overview 4-140
program 5-178
wire 4-134
serial
cable wiring 13-366
configure port for ASCII 13-367
connect an ASCII device 13-366
SFC_ACTION structure 4-138
SFC_STEP structure 4-128
SFC_STOP structure 4-160
SFP instruction 4-163
SFR instruction 4-159, 4-161, 4-163
sheet
add 8-245
connect 8-250
function block diagram 8-231
shut down the controller 16-409
simultaneous branch 4-134
enter 5-171
force 15-393, 15-395
step through 15-393
source key 19-431
status
force 15-389
memory 21-463
monitor 18-421, 18-422
status flags 18-421
step
add action 5-180
alarm 4-145
assign preset time 5-176
configure 5-176
configure alarm 5-177
data type 4-128
define 4-126
organize in sequential function chart 4-131
selection branch 4-133
sequence 4-133
simultaneous branch 4-134

timer 4-145
step through
simultaneous branch 15-393
transition 15-393
stop
data type 4-160
sequential function chart 4-158
store
action 4-155
project 20-451
string
compare characters 14-379, 14-382
convert characters 14-383
create 14-385
data type 13-370
enter characters 13-375
evaluation in structured text 6-198
extract characters 14-378
manipulate 14-377
organize data 13-370
read characters 13-371
search an array of characters 14-379
write characters 13-373
string data type
create 13-370
structure
create 2-46
organize 2-36
overview 2-33
SFC_ACTION 4-138
SFC_STEP 4-128
SFC_STOP 4-160
user-defined 2-44
structured text
arithmetic operators 6-196
assign ASCII character 6-194
assignment 6-192
bitwise operators 6-200
CASE 6-206
comments 5-185, 6-218
components 6-191
contructs 6-202
evaluation of strings 6-198
expression 6-194
FOR…DO 6-209
force a value 15-387
functions 6-196
IF...THEN 6-203
in action 5-182
logical operators 6-199
non-retentive assignment 6-193
numeric expression 6-194
relational operators 6-197
REPEAT…UNTIL 6-215
WHILE…DO 6-212
suspend
controller 16-409
symbol. See alias.
synchronize
controllers 3-101
system data
access 18-422
system overhead time slice 3-115
guidelines for multiple tasks 3-65
impact on executon 3-63

T

tag
address 2-50
alias 2-51
array 2-39
assign 7-227
function block diagram 8-248
assign dimensions 2-43
choose name 7-224, 8-233
consume 11-344
create 2-39, 7-227
create alias 2-53
data type 2-33
description 2-48
enter 7-227
force 15-391, 15-392
guidelines 2-36
guidelines for messages 11-352
I/O 1-27
memory allocation 2-33
name 2-35
organize 2-36
organize for message 11-347
organize produced and consumed tags 11-341
overview 2-31
produce 11-343
produce and consume 11-339
reuse of name 2-35
scope 2-35
string 13-370
trigger event task 3-98
type 2-32
task
assign priority 3-62
avoid overlap 3-66
choose event trigger 3-77
choose type 3-60
create event 3-109

create periodic 3-110
define 4-125
define timeout 3-111
impact of multiple tasks on communication 3-65
inhibit 3-74
manually check for overlap 3-67
manually configure output processing 3-72
monitor 3-67, 3-68
number supported 3-61
output processing 3-70
overlap 3-66
priority 3-62
programmatically check for overlap 3-68
programmatically configure output processing 3-73
trigger via EVENT instruction 3-106
watchdog time 3-118
test a fault routine 16-408
text box
sequential function chart 5-186
show or hide in sequential function chart 5-187
throughput
estimate for event task 3-85
timeout
define for event task 3-111
transition
BOOL expression 4-143
call subroution 5-179
choose program method 4-143
define 4-140
EOT instruction 4-144
force 15-393, 15-395
program 5-178
step through 15-393
use of a subroutine 4-144
trigger
axis registration 3-90
axis watch 3-94
choose for event task 3-77
consumed tag 3-98
EVENT instruction 3-106
module input data 3-79
motion group 3-88
supported by controller 3-78

U

unresolved loop
function block diagram 8-238
user protocol
configure for ASCII 13-368
user-defined data type
create 2-46
guidelines 2-46
overview 2-44

V

verify
routine 5-189, 7-229, 8-251

W

watch point
trigger event task 3-94
watchdog time 3-118
weight
convert 14-383
WHILE…DO 6-212
wire
function block diagram 8-235, 8-238, 8-246
sequential function chart 4-134, 5-175
write
ASCII characters 13-373

ASCII Character Codes

Character	Dec	Hex
[ctrl-@] NUL	0	$00
[ctrl-A] SOH	1	$01
[ctrl-B] STX	2	$02
[ctrl-C] ETX	3	$03
[ctrl-D] EOT	4	$04
[ctrl-E] ENQ	5	$05
[ctrl-F] ACK	6	$06
[ctrl-G] BEL	7	$07
[ctrl-H] BS	8	$08
[ctrl-I] HT	9	$09
[ctrl-J] LF	10	$I ($0A)
[ctrl-K] VT	11	$0B
[ctrl-L] FF	12	$0C
[ctrl-M] CR	13	$r ($0D)
[ctrl-N] SO	14	$0E
[ctrl-O] SI	15	$0F
[ctrl-P] DLE	16	$10
[ctrl-Q] DC1	17	$11
[ctrl-R] DC2	18	$12
[ctrl-S] DC3	19	$13
[ctrl-T] DC4	20	$14
[ctrl-U] NAK	21	$15
[ctrl-V] SYN	22	$16
[ctrl-W] ETB	23	$17
[ctrl-X] CAN	24	$18
[ctrl-Y] EM	25	$19
[ctrl-Z] SUB	26	$1A
ctrl-[ESC	27	$1B
[ctrl-\] FS	28	$1C
ctrl-] GS	29	$1D
[ctrl-^] RS	30	$1E
[ctrl-_] US	31	$1F

Character	Dec	Hex
SPACE	32	$20
!	33	$21
“	34	$22
#	35	$23
$	36	$24
%	37	$25
&	38	$26
‘	39	$27
(	40	$28
)	41	$29
*	42	$2A
+	43	$2B
,	44	$2C
-	45	$2D
.	46	$2E
/	47	$2F
0	48	$30
1	49	$31
2	50	$32
3	51	$33
4	52	$34
5	53	$35
6	54	$36
7	55	$37
8	56	$38
9	57	$39
:	58	$3A
;	59	$3B
<	60	$3C
=	61	$3D
>	62	$3E
?	63	$3F

Character	Dec	Hex
@	64	$40
A	65	$41
B	66	$42
C	67	$43
D	68	$44
E	69	$45
F	70	$46
G	71	$47
H	72	$48
I	73	$49
J	74	$4A
K	75	$4B
L	76	$4C
M	77	$4D
N	78	$4E
O	79	$4F
P	80	$50
Q	81	$51
R	82	$52
S	83	$53
T	84	$54
U	85	$55
V	86	$56
W	87	$57
X	88	$58
Y	89	$59
Z	90	$5A
[	91	$5B
\	92	$5C
]	93	$5D
^	94	$5E
_	95	$5F

Character	Dec	Hex
‘	96	$60
a	97	$61
b	98	$62
c	99	$63
d	100	$64
e	101	$65
f	102	$66
g	103	$67
h	104	$68
i	105	$69
j	106	$6A
k	107	$6B
l	108	$6C
m	109	$6D
n	110	$6E
o	111	$6F
p	112	$70
q	113	$71
r	114	$72
s	115	$73
t	116	$74
u	117	$75
v	118	$76
w	119	$77
x	120	$78
y	121	$79
z	122	$7A
{	123	$7B
|	124	$7C
}	125	$7D
~	126	$7E
DEL	127	$7F

Rockwell Automation Support

Rockwell Automation provides technical information on the Web to assist you in using its products. At http://support.rockwellautomation.com, you can find technical manuals, a knowledge base of FAQs, technical and application notes, sample code and links to software service packs, and a MySupport feature that you can customize to make the best use of these tools.

For an additional level of technical phone support for installation, configuration, and troubleshooting, we offer TechConnect Support programs. For more information, contact your local distributor or Rockwell Automation representative, or visit http://support.rockwellautomation.com.

Installation Assistance

If you experience a problem with a hardware module within the first 24 hours of installation, please review the information that's contained in this manual. You can also contact a special Customer Support number for initial help in getting your module up and running.

United States	1.440.646.3223 Monday – Friday, 8am – 5pm EST
Outside United States	Please contact your local Rockwell Automation representative for any technical support issues.

New Product Satisfaction Return

Rockwell tests all of its products to ensure that they are fully operational when shipped from the manufacturing facility. However, if your product is not functioning, it may need to be returned.

United States	Contact your distributor. You must provide a Customer Support case number (see phone number above to obtain one) to your distributor in order to complete the return process.
Outside United States	Please contact your local Rockwell Automation representative for return procedure.

www.rockwellautomation.com

Power, Control and Information Solutions Headquarters

Americas: Rockwell Automation, 1201 South Second Street, Milwaukee, WI 53204-2496 USA, Tel: (1) 414.382.2000, Fax: (1) 414.382.4444
Europe/Middle East/Africa: Rockwell Automation, Vorstlaan/Boulevard du Souverain 36, 1170 Brussels, Belgium, Tel: (32) 2 663 0600, Fax: (32) 2 663 0640
Asia Pacific: Rockwell Automation, Level 14, Core F, Cyberport 3, 100 Cyberport Road, Hong Kong, Tel: (852) 2887 4788, Fax: (852) 2508 1846

Publication 1756-PM001I-EN-P - January 2007 PN 953030-27
Supersedes Publication 1756-PM001H-EN-P - June 2005
Copyright © 2007 Rockwell Automation, Inc. All rights reserved. Printed in the U.S.A.